Dissertação

Mestrado em Engenharia Informática – Computação Móvel

Monitorização de Obras Marítimas e Portuárias

Carlos Jorge Machado Antunes

Leiria, Fevereiro de 2011

Dissertação

Mestrado em Engenharia Informática – Computação Móvel

Monitorização de Obras Marítimas e Portuárias

Carlos Jorge Machado Antunes

Dissertação de Mestrado realizada sob a orientação do Doutor Patrício Rodrigues Domingues, Professor da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria.

Leiria, Fevereiro de 2011

À Família e Amigos

Esta página foi intencionalmente deixada em branco

Agradecimentos

Aproveito este espaço para falar na primeira pessoa e deixar uma palavra de agradecimento a todos aqueles que, de uma forma ou de outra, contribuíram para a conclusão deste projecto. Agradeço:

- Em primeiro lugar, a Deus, por me ter dado uma oportunidade de andar nesta Terra;
- À minha família, especialmente à minha mãe, por me ter ensinado a andar;
- Ao meu orientador, o Professor Doutor Patrício Domingues, por me ter ajudado a escolher o caminho a seguir;
- À Sara, por acompanhar-me nesta caminhada;
- Ao Eng.º Rodrigues Vieira e Jorge Caldeira, por todo o apoio dado nesta fase do meu trajecto;
- E por fim, mas não por último, ao Pedro Catarino, meu companheiro de armas, que aparece no caminho sempre que é preciso.

Muito Obrigado!

Esta página foi intencionalmente deixada em branco

Resumo

A tomada de decisão respeitante à realização de obras de manutenção e/ou reparação de uma obra marítima é apoiada com base no diagnóstico obtido a partir da realização de um conjunto de inspecções visuais periódicas, enquadradas num programa de acompanhamento da evolução da obra previamente estabelecido. Estas estruturas têm normalmente um grande comprimento e, por isso, para a realização da inspecção considera-se que são "decompostas" em troços pequenos (com comprimentos de 50 a 80 metros), identificados pelas suas coordenadas limites, que se manterão no tempo e que são caracterizados sistematicamente de forma independente em relação aos troços limites. Actualmente, o resultado de uma inspecção é registado em formato de papel através do preenchimento de um formulário por cada troço observado, onde se atribuem os valores da avaliação dos critérios estabelecidos para o tipo de obra em análise.

O principal objectivo deste projecto consistiu em criar um sistema capaz de gerir, de forma integrada, a realização de inspecções visuais a obras marítimas, sendo composto por:

- Uma aplicação *desktop* que permite registar uma ou mais obras marítimas e definir os formulários de inspecção respectivos;
- Um Web Service que disponibiliza pela rede os dados dessas obras e formulários de inspecção, como também permite o envio dos dados dos formulários preenchidos;
- Uma aplicação móvel que permite realizar o preenchimento dos formulários de inspecção no local da obra.

Com este sistema, foi possível eliminar a necessidade da utilização de papel e optimizar a disponibilidade dos dados relativos às inspecções.

Palavras-chave: estrutura, marítima, monitorização, inspecção, móvel

Esta página foi intencionalmente deixada em branco

Abstract

The decision making concerning maintenance work and/or repair of maritime structures is supported based on diagnoses obtained from the execution of a series of periodic visual inspections, under a previously established program for monitoring the evolution of a structure. These structures are usually very long and, therefore, are considered to be "broken down" into small parts (with lengths from 50 to 80 meters), that can be easily identified by its coordinates limits, which will remain constant in time and are systematically characterized independently in relation to its adjacent parts. Currently, the result of an inspection is written in paper forms by filling in a form for each part observed, in which the inspector assigns values based on the evaluation criteria established for the type of structure in analysis.

The main objective of this project was to create a system with the ability to define and carry out visual inspections of maritime structures, being composed by:

- A desktop application that allows to register one or more maritime structures and to create their visual inspection forms;
- A Web service that provides through the network data regarding these structures and inspection forms and also allows receiving data of the filled out forms;
- A mobile application that allows to fill in the inspection forms of the structure on-site.

With this system, it's possible to eliminate the need for paper forms and maximize the availability of data related to the inspections.

Key-Words: structure, maritime, monitoring, inspection, mobile

Esta página foi intencionalmente deixada em branco

Índice de Figuras

Figura 1- Exemplos de obras marítimas do tipo quebra-mar.....5

Figura 2 - Decomposição de obras marítimas em troços.....7

Figura 3 - Formulário inicial da aplicação ANOSOM.....11

Figura 4 - DER de alto nível da BD da ANOSOM.....11

Figura 5 - Janela de visualização de informação geográfica da SANDS.....13

Figura 6 - DER de alto nível do OpenStreetMap.....20

Figura 7 - Percentagem de mercado de dispositivos móveis por região.....24

Figura 8 - Percentagem de mercado e crescimento por fabricante.....25

Figura 9 - Percentagem de mercado e crescimento por sistema operativo.....26

Figura 10 - Visão geral do sistema.....33

Figura 11 - Estruturação de uma aplicação por camadas.....34

Figura 12 - Visão geral dos módulos da aplicação desktop.....35

Figura 13 - Diagrama de classes do módulo Inversion of Control.....39

Figura 14 - A classe ModuleInstall.....40

Figura 15: Diagrama de classes do padrão de desenho Repository.....41

Figura 16 - Diagrama de classes do padrão de desenho Unit of Work.....42

Figura 17 - Diagrama de classes do modelo de entidades.....44

Figura 18 - Diagrama de classes da implementação do padrão de desenho MVC.....46

Figura 19 - Descrição dos botões da barra de edição.....47

Figura 20 - Diagrama de classes do modelo de navegação.....48

Figura 21 - Descrição dos botões da barra de navegação.....48

Figura 22 - Diagrama de classes do modelo de gestão de utilizadores.....49

Figura 23 - Definição e implementação da interface DataSourceDescriptor.....50

Figura 24 - Janela de autenticação e de configuração da ligação ao servidor.....50

Figura 25 - Diagrama de classes do modelo de geometrias.....51

Figura 26 - Diagrama de classes do modelo de fichas de inspecção.....52

Figura 27 - Diagrama de classes do modelo de fichas de inspecção preenchidas.....53

Figura 28 - Janela de gestão de fichas de inspecção.....54

Figura 29 - Modelo de PropertyEditors para colecções de objectos.....55

Figura 30 - Janela de gestão de respostas a uma pergunta.....56

Figura 31 - Diagrama de classes do modelo de obras marítimas.....57

Figura 32 - Janela de gestão de estruturas de um porto.....57

Figura 33 - Janela de visualização de fichas de inspecção preenchidas.....58

Figura 34 - Visão geral dos módulos da aplicação servidor.....59

Figura 35 - Diagrama de classes do padrão de desenho Entity Translator.....61

Figura 36 - Implementação do padrão de desenho Entity Translator.....63

Figura 37 - Diagrama de classes do modelo de mensagens.....64

Figura 38 - Transmissão de mensagens entre cliente móvel e Web Service.....65

Figura 39 - Diagrama de sequência do processamento de um pedido de dados.....66

Figura 40 - Diagrama de sequência da persistência de dados na aplicação servidor.....66

Figura 41 - Visão geral dos módulos da aplicação móvel.....67

Figura 42 - Diagrama de classes do módulo Data.....68

Figura 43 - Diagrama de classes do modelo de domínio da aplicação móvel.....69

Figura 44 -Diagrama de classes da implementação do MVC.....70

Figura 45 - Fluxo de janelas da aplicação móvel.....71

Figura 46 - Diagrama de classes das Views e Controllers da aplicação móvel.....72

Figura 47 - Particionamento de uma ficha de inspecção em páginas.....73

Figura 48 - Diagrama de classes da interface gráfica das fichas de inspecção.....74

Figura 49 - Exemplo do preenchimento de uma ficha de inspecção.....74

Figura 50 - Diagrama de classes dos serviços da aplicação móvel.....75

Figura 51 - Diagrama de sequência da persistência de dados na aplicação cliente.....76

Figura 52 - Diagrama de sequência do processo de envio de dados.....77

Esta página foi intencionalmente deixada em branco

Índice de Quadros

Tabela 1 - Caracterização técnica de obras marítimas.....6

Tabela 2 - Ficha de inspecção do manto resistente.....8

Tabela 3 - Ficha de inspecção de paramentos.....9

Tabela 4- Requisitos gerais das aplicações.....15

Tabela 5 - Temas sobre TIs definidos para investigação.....16

Tabela 6 - Plataformas para desenvolvimento de aplicações.....22

Tabela 7 - Ferramentas SIG existentes no mercado.....23

Tabela 8 - Encomendas de dispositivos móveis por região.....24

Tabela 9: Encomendas de dispositivos móveis por fabricante.....25

Tabela 10 - Encomendas de dispositivos móveis por sistema operativo.....26

Tabela 11 - Fabricantes de dispositivos móveis para fins empresariais.....27

Tabela 12 - Comparação entre SGBDs para dispositivos móveis.....28

Tabela 13 - SIG's para dispositivos móveis.....29

Tabela 14 - Mapeamento entre o tipo de resposta e componentes gráficos.....72

Esta página foi intencionalmente deixada em branco

Lista de Siglas

ANOSOM – Análise de Observação Sistemática de Obras Marítimas

API – Application Programming Interface

BD – Base de Dados

CDDL – Common Development and Distribuition License

CPU – Central Processing Unit

DDD – Domain-Driven Design

DER – Diagrama Entidade-Relacionamento

EJB – Enterprise JavaBean

EPL – Eclipse Public License

ERP – Enterprise Resource Planning

GPL – General Public License

GPS – Global Positioning System

HTTP – HyperText Transfer Protocol

IDE – Integrated Development Environment

ISV – Independent Software Vendor

J2EE – Java 2, Enterprise Edition

J2SE – Java 2, Standard Edition

JDK – Java Development Kit

JPA – Java Persistence API

JTS – Java Topology Suite

LGPL – Lesser GPL

LNEC – Laboratório Nacional de Engenharia Civil

MPL – Mozilla Public License

Ms-PL – Microsoft Public License

MVC – Model-View-Controller

O/RM – Object-Relational Mapping

PC – Personal Computer

RAM – Random Access Memory

RCP – Rich Client Platform

SANDS – Shoreline And Nearshore Data System

SDK – Software Development Kit

SIG – Sistemas de Informação Geográfica

SGBD – Sistemas de Gestão de Base de Dados

SOAP – Simple Object Access Protocol

SQL – Structured Query Language

SWT – Standard Widget Toolkit

TI – Tecnologias de Informação

UI – User Interface

VBA – Visual Basic for Applications

XML – eXtended Markup Language

Esta página foi intencionalmente deixada em branco

Índice

AGRADECIMENTOS ..... **III**

RESUMO ..... **V**

ABSTRACT ..... **VII**

ÍNDICE DE FIGURAS ..... **IX**

ÍNDICE DE QUADROS ..... **XIII**

LISTA DE SIGLAS ..... **XV**

ÍNDICE ..... **XVII**

1.INTRODUÇÃO **1**

1.1.Motivação 2
1.2.Objectivos 2
1.3.Estrutura do documento 3

2.REVISÃO DA LITERATURA **5**

2.1.Caracterização de obras marítimas 5
2.1.1.Critérios de observação de obras marítimas *7*
2.1.2.Avaliação da importância dos danos *9*
2.2.Aplicações de monitorização de obras marítimas 10
2.2.1.ANOSOM *10*
2.2.1.1.Base de Dados da ANOSOM 11
2.2.2.SANDS *12*
2.2.3.Conclusões *14*
2.3.Tecnologias de Informação 15
2.3.1.SGBDs com suporte para dados geográficos *16*
2.3.2.Bibliotecas de gestão de dados numa BD *18*
2.3.2.1.OpenStreetMap 19
2.3.3.Plataformas para desenvolvimento de aplicações desktop *21*
2.3.4.Ferramentas SIG *22*
2.3.5.Dispositivos móveis existentes no mercado *23*
2.3.6.Dispositivos móveis para fins empresariais *26*
2.3.7.Linguagens de programação e bibliotecas para aplicações móveis *27*
2.3.8.SGBDs para aplicações móveis *28*
2.3.9.Ferramentas SIG para dispositivos móveis *28*
2.3.10.Servidores aplicacionais *29*
2.3.11.Conclusões *30*

3.ARQUITECTURA **33**

3.1.Arquitectura da aplicação desktop 35
3.1.1.Inversion Of Control *37*
3.1.2.Data *40*

3.1.3.Data UI.....*45*
3.1.4.Membership.....*49*
3.1.5.Geometry Model.....*51*
3.1.6.Inspections Model.....*51*
3.1.7.Structures Model.....*56*
3.2.Arquitectura da aplicação servidor.....58
3.2.1.Entity Translator (Tradutor de Entidades).....*60*
3.2.2.Descrição detalhada da comunicação entre aplicações.....*65*
3.3.Arquitectura da aplicação móvel.....67
3.3.1.Data.....*67*
3.3.2.Model.....*69*
3.3.3.UI.....*70*
3.3.4.Services.....*75*
4.CONCLUSÃO.....79
4.1.Desenvolvimentos futuros.....82
BIBLIOGRAFIA.....83
ANEXOS.....85

1. Introdução

A monitorização de obras marítimas e portuárias é um processo que permite medir e quantificar o eventual estado de degradação de uma obra que é, em geral, o resultado de um processo lento de erosão. A monitorização tem como base um programa de inspecções sistemático, contínuo e de longo prazo que inclui inspecções periódicas, de rotina e especiais (em caso de acidente ou outras anomalias). As inspecções podem ser realizadas de várias formas, de acordo com as técnicas de medição utilizadas.

O tema deste projecto enquadra-se nas inspecções por observação visual. A observação visual expedita, para além de pouco onerosa em comparação com outras técnicas (por exemplo, a fotogrametria), permite identificar a maior parte dos problemas e avaliar a importância dos mesmos, desde que feita regularmente. Tem a contrapartida de envolver alguma subjectividade nas observações, o que pode ser minimizado pelo estabelecimento de critérios tanto quanto possível objectivos de avaliação e quantificação dos estragos detectados.

As inspecções visuais são efectuadas por técnicos especializados que avaliam o estado dos vários elementos estruturais de uma obra. Os dados da avaliação são registados recorrendo a um conjunto de formulários em papel específicos para cada tipo de elemento estrutural. Os dados recolhidos são posteriormente comparados com os dados de inspecções anteriores, de forma a averiguar se é necessário efectuar obras de manutenção ou de reparação. Esta comparação é feita com base em critérios de classificação de danos definidos para cada tipo de obra.

O objectivo principal deste projecto foi criar um sistema através do qual é possível gerir, de forma integrada, a realização de inspecções visuais de obras marítimas. Este projecto é parte

integrante de outro projecto real de maiores dimensões, levado a cabo por uma empresa de consultores de hidráulica e obras marítimas, cujo âmbito abrange inspecções a outros tipos de obras e activos, nomeadamente, obras portuárias, qualidade da água, praias, entre outros.

1.1. Motivação

Actualmente, o resultado de uma observação visual expedita é registado em formato de papel através do preenchimento de um formulário por elemento estrutural de cada troço observado, onde se atribuem os valores da avaliação dos critérios estabelecidos para o tipo de obra em análise. Os dados recolhidos são posteriormente introduzidos através de uma aplicação *desktop* a partir da qual é possível analisar as fichas de inspecção preenchidas, ou seja, avaliar as respostas dadas pelo inspector, de modo a determinar se é necessário realizar obras de manutenção e/ou reparação.

As principais motivações para o desenvolvimento deste projecto prendem-se com:

- A necessidade de eliminar a utilização de papel;
- A optimização e agilização da introdução e obtenção de dados de inspecções realizadas, através da redução da necessidade de intervenção humana, diminuindo assim a probabilidade da ocorrência de erros;
- A optimização da disponibilidade dos dados das fichas de inspecção preenchidas;
- A disponibilização, de forma acessível e centralizada, de dados históricos de todas das inspecções realizadas, de modo a ser possível analisar a evolução de uma obra.

1.2. Objectivos

O objectivo principal deste trabalho é criar um sistema que permita:

- Caracterizar uma ou mais obras marítimas, tendo em conta os seus elementos estruturais e decomposição por troços;
- Definir uma ou mais fichas de inspecção associadas aos elementos estruturais de uma obra marítima;
- Assinalar uma ou mais obras marítimas para inspecção;

- Disponibilizar remotamente os dados relativos às obras assinaladas para inspecção e respectivas fichas de inspecção;
- Realizar o preenchimento das fichas de inspecção no local da obra marítima, recorrendo, para isso, a um dispositivo móvel;
- Recolher os dados das fichas de inspecção preenchidas para posterior avaliação.

1.3. Estrutura do documento

O documento está estruturado em quatro capítulos que focam os seguintes aspectos:

- **Introdução** – Faz uma breve descrição do tema em estudo, identificando as motivações e os objectivos atingidos com este projecto;
- **Revisão da literatura** – Dedicado à apresentação da informação relevante recolhida acerca da monitorização de obras marítimas e Tecnologias de Informação (TI);
- **Arquitectura** – Apresenta uma descrição detalhada da arquitectura das aplicações desenvolvidas no âmbito deste projecto. Contém três secções dedicadas a cada aplicação – *desktop*, servidor e móvel;
- **Conclusões** – Descreve sumariamente os desenvolvimentos realizados no contexto deste projecto, apresentando uma análise crítica sobre os trabalhos realizados e sugestões para desenvolvimentos futuros.

Esta página foi intencionalmente deixada em branco

2. Revisão da literatura

Este capítulo apresenta informação relevante recolhida acerca da monitorização de obras marítimas e tecnologias de informação que se enquadrem no âmbito deste projecto.

2.1. Caracterização de obras marítimas

Considera-se que obra marítima é qualquer estrutura ou conjunto de estruturas executadas com materiais de vida útil longa (superior a 20 anos), existentes na faixa da orla costeira e que está potencialmente sujeita à acção directa da agitação marítima. A sua caracterização técnica pode ser feita com base em:

- Elementos estruturais;
- Materiais;
- Dimensões;

Figura 1- Exemplos de obras marítimas do tipo quebra-mar

A tabela 1 apresenta sucintamente algumas das classificações existentes para obras marítimas e para elementos estruturais, referindo também alguns exemplos de obras de acordo com o seu tipo.

<table>
<tr><th colspan="3">Caracterização técnica de obras marítimas</th></tr>
<tr><th colspan="2">Classificação</th><th>Elementos estruturais</th></tr>
<tr><th>Tipo</th><th>Exemplos</th><td rowspan="6">Infra-estruturas:
• Núcleo de quebra-mar
• Manto resistente
• Caixotão
Super-estruturas:
• Maciço de coroamento
• Muro cortina
• Corta águas
• Paramentos</td></tr>
<tr><td>Defesa</td><td>Esporões
Revestimentos</td></tr>
<tr><td>Abrigo</td><td>Molhes
Quebra-mares</td></tr>
<tr><td>Regularização e correcção hidráulica</td><td>Diques
Esporões</td></tr>
<tr><td>Reparação e manutenção naval</td><td>Planos inclinados
Grades de marés</td></tr>
<tr><td>Terrestres</td><td>Terraplenos
Pavimentos</td></tr>
</table>

Tabela 1 - Caracterização técnica de obras marítimas

Dado que estas obras têm normalmente um grande comprimento, para efeitos da monitorização, a sua estrutura é tipicamente decomposta em troços de pequeno tamanho, identificados pelas suas coordenadas limites e caracterizados sistematicamente de forma independente em relação aos troços limites. A avaliação da necessidade de realização das eventuais obras de manutenção ou reparação, o chamado diagnóstico da estrutura, deve reflectir a evolução da estrutura desde uma data considerada relevante para o efeito. Este diagnóstico é feito com base na informação recolhida em campanhas de inspecção visual, nas quais são preenchidos formulários de inspecção, um para cada elemento de cada troço da estrutura, onde se caracterizam os vários componentes do perfil do troço. A figura 2 mostra uma imagem do Porto da Nazaré que exemplifica a decomposição de duas obras (os molhes norte e sul) por troços.

Figura 2 - Decomposição de obras marítimas em troços

2.1.1. Critérios de observação de obras marítimas

Os critérios de observação são estabelecidos tendo em conta o tipo de estruturas a observar e as dificuldades que essa observação envolve. É também necessário ter em consideração que uma inspecção desta natureza envolve alguma subjectividade, o que pode ser minimizado através do estabelecimento de critérios de avaliação, tanto quanto possível objectivos, e da quantificação dos estragos detectados. Por exemplo, para um quebra-mar a observação de cada troço deverá avaliar aspectos que condicionam a resistência da estrutura, tais como:

- Existência e quantidade de quedas de blocos;
- Existência, quantidade e aspecto superficial das fracturas de blocos;
- Estado geral do talude (continuidade e assentamento);
- Existência ou não de degradação do material dos blocos;
- Passagem "anormal" de energia através da estrutura;

A qualquer destes aspectos é atribuído um grau (por exemplo, inexistência de degradação, degradação pequena, média ou grande). Para facilitar a caracterização, é indicado um valor de danos correspondente a cada grau (no caso das quedas ou das fracturas) ou uma designação sugestiva (no caso do estado geral do talude ou da degradação do material). O estado do material constituinte dos blocos é ainda avaliado pelo estado das aresta dos elementos estruturais (que, seja por abrasão, seja por corrosão química, tendem a arredondar).

Todos os dados recolhidos da inspecção são registados em formulários desenvolvidos especificamente para cada tipo de elemento estrutural a observar. As tabelas 2 e 3 são exemplos de fichas de inspecção que mostram os dados que devem ser recolhidos para o manto resistente e paramento, respectivamente.

MANTO RESISTENTE							
Quedas de Blocos		**Fracturas de Blocos**					
		Quantidade		**Aspecto Superficial**			
Muitas (+ 10 blocos)		Muitas (+ 10 blocos)		Em concha			
Algumas (5 a 10)		Algumas (5 a 10)		Irregular			
Poucas (1 a 5)		Poucas (1 a 5)		Plana			
Nenhumas		Nenhumas		Outro			
Talude		**Degradação Superficial do Material dos Blocos**					
		Quantidade		**Descrição**			
Em bom estado		Em bom estado		Cantos intactos			
Degradado junto à linha de água		Bom, mas com muitos poros superficiais		Cantos arredondados			
Degradado		Alguma corrosão		Som	Sólido		
Muito Degradado		Muita corrosão			Cavo		
Assentamento do Manto		Junto à linha de água		Estimativa			
		Coroamento		Estimativa			
Passagem de energia através do núcleo		Não ocorre					
		Ocorre		Pequena Intensidade			
				Média Intensidade			
				Grande Intensidade			

Tabela 2 - Ficha de inspecção do manto resistente

PARAMENTO / SUPERESTRUTURA							
Fracturas				**Degradação superficial do material**			
Quantidade		**Continuidade**		**Grau**		**Caracterização**	
Muitas (+ de 10)		Grande (+ de 2 m)		Em bom estado		De origem física	
Algumas (6 a 10)		Média (1 a 2 m)		Pequena		De origem química	
Poucas (1 a 5)		Pequena (até 1 m)		Média		Arredondamento	
Nenhumas				Grande		Recobrimento danificado	
						Corrosão de armaduras	
Infraescavação				**Assentamento**			
Não existe		**Ordem de grandeza**		Não existe		**Ordem de grandeza**	
Existe				Existe			
		Amplitude				Amplitude	
		Extensão				Extensão	
Deslizamento				Derrubamento			
Não existe		Ordem de grandeza		Não existe		Ordem de grandeza	
Existe				Existe			
		Amplitude				Amplitude	
		Extensão				Extensão	

Tabela 3 - Ficha de inspecção de paramentos

2.1.2. Avaliação da importância dos danos

Após a realização das inspecções torna-se necessário aferir a importância das anomalias detectadas, com vista a estabelecer os procedimentos a adoptar. Para essa avaliação será sempre importante a contribuição da experiência da equipa que faz a inspecção. No entanto, torna-se necessário definir critérios que minimizem a subjectividade das conclusões. Não há critérios gerais de avaliação da importância dos estragos neste tipo de obras. Esses critérios são diferentes em cada tipo de estrutura e estão relacionados com a importância de cada elemento estrutural e com o funcionamento global da estrutura. Na tabela A.1 dos anexos apresenta-se a classificação de danos de uma estrutura de taludes, do tipo quebra-mar. Os danos são classificados de forma crescente com a importância em 4 grandes grupos, atribuíveis com base na importância dos vários aspectos observados. São também indicados os riscos de ruína associados àqueles grupos, bem como o tipo de programação de inspecções

(de rotina ou extraordinárias) e de intervenções de manutenção e/ou reparação a efectuar. A tabela A.2 dos anexos apresenta outro exemplo de classificação de danos para paramentos.

2.2. Aplicações de monitorização de obras marítimas

Nesta secção apresenta-se um levantamento do estado de arte no que respeita a aplicações já existentes relativas à monitorização de obras marítimas.

2.2.1. ANOSOM

Em Portugal, foi desenvolvida uma aplicação pelo Laboratório Nacional de Engenharia Civil (LNEC) denominada ANOSOM (Análise de Observação Sistemática de Obras Marítimas), para monitorização de obras marítimas que permite efectuar o diagnóstico do estado de conservação de obras marítimas do tipo quebra-mar [1][2]. Implementada recorrendo ao Microsoft Office Access, esta aplicação permite realizar um conjunto de tarefas que se enquadram em quatro categorias:

- Gestão de dados referentes às estruturas, nos quais se incluem os relativos à divisão conceptual das mesmas em troços, a caracterização dos elementos da envolvente do perfil de cada um desses troços e as datas da realização de obras de reparação ou manutenção nos mesmos troços;

- Introduzir dados de campanhas de inspecção visual, nos quais se incluem os conteúdos dos formulários de inspecção, assim como fotografias tiradas no decorrer das campanhas;

- Introdução de dados de levantamentos da envolvente da estrutura, contendo as coordenadas dos pontos levantados;

- Realização de um conjunto de análises que avaliam a necessidade de realização de obras de manutenção ou reparação.

A figura 3 mostra o formulário inicial da aplicação ANOSOM, onde é possível observar mais detalhadamente o conjunto de funcionalidades disponibilizadas.

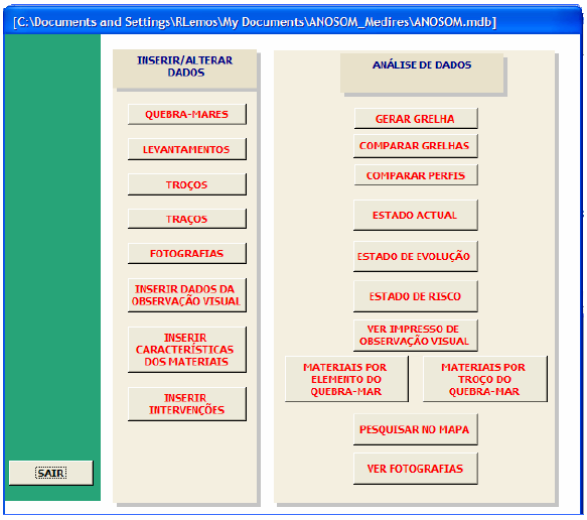

Figura 3 - Formulário inicial da aplicação ANOSOM

2.2.1.1. Base de Dados da ANOSOM

A Base de Dados (BD) da aplicação ANOSOM foi pensada para armazenar os dados relativos a estruturas do tipo quebra-mar, incluindo os seus troços e materiais constituintes, para além dos dados das intervenções e inspecções visuais realizadas. Torna-se, por isso, relevante apresentar uma explicação mais detalhada acerca da estrutura da BD de modo a facilitar a compreensão das funcionalidades da aplicação. A figura 4 mostra o Diagrama Entidade-Relacionamento (DER) de alto nível da BD da ANOSOM, fornecendo uma visão geral das tabelas e relacionamentos definidos.

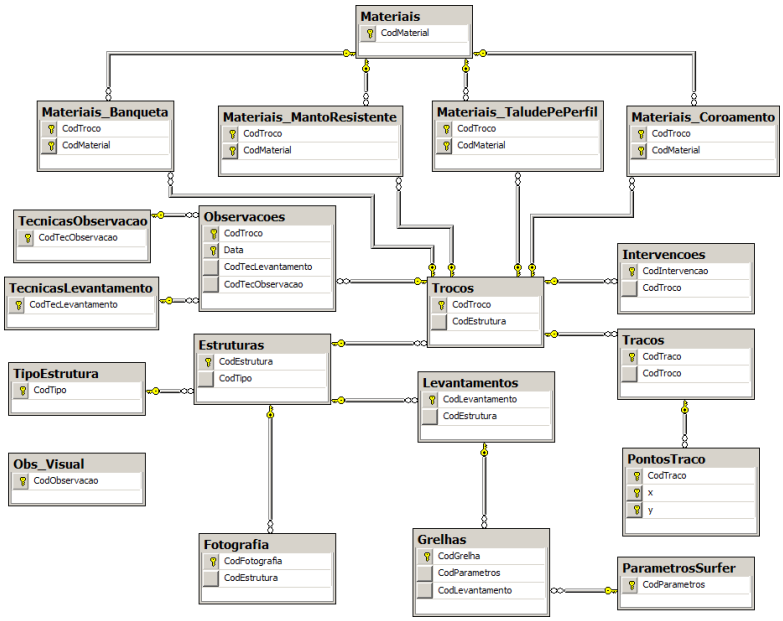

Figura 4 - DER de alto nível da BD da ANOSOM

De seguida, descreve-se o papel das tabelas mais relevantes:

- **Materiais** – Contém os registos dos materiais utilizados na construção de qualquer estrutura, assim como as suas principais características físicas;
- **Estruturas** – Guarda a informação relativa a estruturas do tipo quebra-mares, tal como o seu nome, localização e ano de construção;
- **Trocos** – Regista toda a informação relativa à caracterização física de cada um dos troços de cada quebra-mar;
- **Fotografia** – Encontram-se compiladas as fotografias consideradas relevantes para a apreciação do estado geral do quebra-mar, para cada campanha de observação e para cada troço. Os ficheiros referentes às fotografias ficam guardados numa pasta local, relativa à aplicação, e nesta tabela apenas são guardados os caminhos dos ficheiros. Para além disto, é também armazenado, para cada registo, a coordenada onde a fotografia foi obtida, assim como a orientação do fotógrafo (virado para a cabeça ou para o enraizamento da estrutura);
- **Intervencoes** – Guarda a informação relativa às obras que determinado troço sofreu ao longo da sua vida útil. Uma intervenção poderá ocorrer com o objectivo de repor o estado inicial da obra após um temporal ou ainda com o objectivo de alterar a sua configuração e/ou materiais utilizados;
- **TecnicasLevantamento** e **TecnicasObservacao** – Enumeram as técnicas passíveis de serem utilizadas nos levantamentos e campanhas de observação;
- **Levantamentos** – Guarda a informação relativa aos levantamentos efectuados para as diferentes estruturas.

2.2.2. SANDS

A SANDS – Shoreline And Nearshore Data System – é uma aplicação comercial desenvolvida pela Halcrow para gestão de zonas costeiras e áreas portuárias. A informação

que foi possível recolher é a que consta no *site* da Halcrow (http://www.halcrow.com/sands/), onde também se pode visualizar uma demonstração da aplicação em Flash.

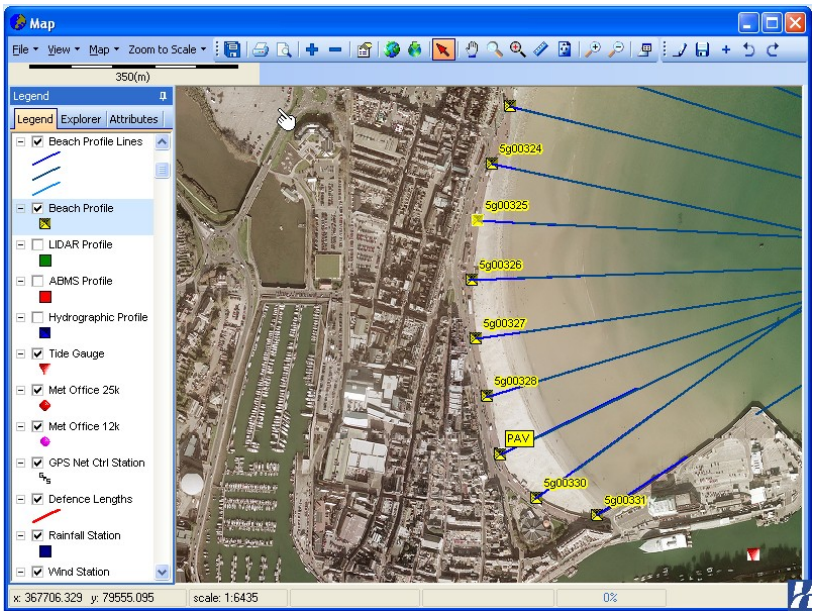

Figura 5 - Janela de visualização de informação geográfica da SANDS

Esta aplicação é composta pelos seguintes módulos:

- **Importação e exportação de dados** – a SANDS tem a capacidade de importar e exportar dados em vários formatos, incluindo séries temporais, registos de marés e ventos, inspecções visuais, entre outros;

- **Aquisição de dados** – Permite efectuar a recolha de dados obtidos a partir de equipamentos dedicados;

- **Gestão de activos** – Possibilita o registo, edição e análise de dados de activos das zonas costeiras e áreas portuárias registadas na aplicação;

- **Mapeamento** – Possui um módulo para gestão de informação geográfica que permite a localização e visualização dos dados referentes às zonas costeiras e áreas portuárias registadas na aplicação. Através deste módulo é possível configurar as áreas de gestão, importar informação georreferenciada e imagens de mapas, desenhar geometrias e realizar medições. As geometrias normalmente representam localizações de interesse,

a partir das quais o utilizador pode consultar os dados alfa-numéricos associados como, por exemplo, as inspecções realizadas até ao momento;

- **Diário de Inspecções** – Permite consultar as fichas de inspecção registadas na aplicação. Estas fichas vêm pré-configuradas para a realização de inspecções a várias tipos de activos, nomeadamente, estruturas portuárias , qualidade da água, praias, corais, entre outros;

- **Gráficos** – Módulo de visualização e análise de séries temporais recorrendo a gráficos;

- **Análises** – Tem a capacidade para realizar análises para determinar efeitos de erosão e acreção, efectuar o cálculo de áreas e volumetrias, prever e analisar marés, calcular valores extremos de ventos e ondas, compreender o transporte de sedimentos, entre outros.

2.2.3. Conclusões

Foram descritas as principais funcionalidades de duas aplicações cujas características se assemelham ao sistema que se pretende implementar. De uma forma geral, ambas as aplicações permitem caracterizar uma obra marítima e gerir os dados de inspecções realizadas nestas obras. Importa, contudo, salientar os principais aspectos que diferenciam este projecto das aplicações apresentadas:

- É um sistema composto por várias aplicações que funcionam de forma integrada e flexível, dado que o utilizador pode (re)definir fichas de inspecção e disponibilizá-las num dispositivo móvel;

- O preenchimento de uma ficha de inspecção é realizado recorrendo a um dispositivo móvel e, por isso, elimina a necessidade da utilização de papel;

- É possível disponibilizar de imediato os dados para a realização de inspecções, assim como os dados da fichas de inspecção preenchidas.

2.3. Tecnologias de Informação

Nesta secção apresenta-se um levantamento do estado de arte respeitante a TIs, dando especial ênfase àquelas que se enquadram no âmbito deste projecto. Os temas a investigar foram definidos com base na informação recolhida na secção anterior, principalmente a que diz respeito às aplicações ANOSOM e SANDS. Os temas encontram-se divididos pelas aplicações que compõem o sistema:

- uma aplicação *desktop* para configuração das fichas de inspecção;
- uma aplicação móvel para o preenchimento das fichas;
- uma aplicação servidor para a sincronização de dados entre as duas aplicações.

A tabela 4 apresenta os requisitos que o sistema deve ter, em traços gerais.

Aplicação	Requisitos gerais
Desktop	- Gerir os dados respeitantes às obras marítimas de um ou mais portos, nos quais se incluem dados sobre elementos estruturais e seus materiais constituintes. Os dados incluem informação geográfica; - Definir fichas de inspecção e permitir associar estas fichas a elementos estruturais de uma obra marítima; - Assinalar uma ou mais obras marítimas para inspecção; - Consultar as fichas de inspecção preenchidas; - Avaliar uma ficha de inspecção preenchida;
Móvel	- Obter e armazenar os dados relativos às obras por inspeccionar; - Preencher as fichas de inspecção e armazenar as respostas dadas pelo inspector; - Enviar as fichas preenchidas;
Servidor	- Disponibilizar um serviço para a obtenção de toda a informação relevante para a realização de uma ou mais inspecções; - Disponibilizar um serviço para enviar as fichas de inspecção preenchidas;

Tabela 4- Requisitos gerais das aplicações

Com base nos requisitos referidos anteriormente, os temas definidos para a realização da investigação encontram-se descritos na tabela 5.

Aplicação	Temas
Desktop	- Sistemas de Gestão de Base de Dados (SGBD) com suporte para dados geográficos; - Bibliotecas para gestão de dados numa BD, incluindo dados geográficos; - Plataformas para desenvolvimento de aplicações, também designadas por Enterprise Resource Planning (ERP) ou Rich Client Platforms (RCP); - Ferramentas de Sistemas de Informação Geográfica (SIG) actualmente disponíveis no mercado.
Móvel	- Dispositivos móveis existentes no mercado; - Dispositivos móveis para fins empresariais; - Linguagens de programação e bibliotecas; - SGBDs para dispositivos móveis; - Ferramentas SIG.
Servidor	- Servidores aplicacionais existentes

Tabela 5 - Temas sobre TIs definidos para investigação

2.3.1. SGBDs com suporte para dados geográficos

Existem actualmente vários SGBDs com suporte para dados geográficos. Neste capítulo serão comparados quatro SGBDs desenvolvidos pelos principais fabricantes destes sistemas – Oracle, Microsoft SQL Server, PostgreSQL e MySQL. A comparação será efectuada principalmente com base nos seguintes critérios:

- Características técnicas gerais;
- As funcionalidades de análise espacial;
- As bibliotecas disponibilizadas pelos fabricantes para desenvolvimento de aplicações SIG;

- O preço;

São apresentados nos anexos as tabelas B.1 e B.2 relativas a características gerais e funcionalidades espaciais, que fazem a comparação cruzada entre os SGBDs em análise [3] [4]. Destas tabelas podem ser retiradas as seguintes conclusões:

- Exceptuando o SQL Server, todos os SGBDs são multi-plataforma;
- O PostgreSQL e o MySQL são gratuitos. Os restantes fornecem versões gratuitas, designadas por versões *Express*, e que apresentam limitações no que diz respeito às funcionalidades e à utilização de recursos do computador (CPU, RAM, tamanho em disco da BD, entre outros);
- O Oracle foi pioneiro nos desenvolvimentos de funcionalidades geográficas;
- O PostgreSQL foi o primeiro SGBD gratuito a disponibilizar funcionalidades geográficas;
- O preço do Oracle com suporte completo para dados geográficos ronda os USD $65000 (versão Enterprise + Oracle Spatial). O preço do SQL Server 2008 ronda os USD $58000;
- O MySQL é o único SGBD que não possui uma biblioteca para manipulação de dados geográficos;
- As bibliotecas de manipulação de dados geográficos dos SGBDs Oracle e PostgreSQL estão implementadas em JAVA. Para o SQL Server, as bibliotecas estão implementadas em .NET;
- O Oracle é o único que suporta o processamento de geometrias 3D (por exemplo, cálculo de volumetrias);
- O SQL Server é o único que não tem suporte integrado para a realização de transformações de sistemas de referência;
- O MySQL é o SGBD que possui menos funcionalidades geográficas.

2.3.2. Bibliotecas de gestão de dados numa BD

De uma forma geral, pode afirmar-se que as principais linguagens de programação orientadas a objectos (e não só) fornecem, de algum modo, suporte para acesso a uma BD. Resumidamente, o programador tem à sua disposição uma biblioteca de código que lhe permite estabelecer uma ligação a uma BD, enviar um comando SQL e, dependendo do tipo de comando (por exemplo, um SELECT), obter uma resposta da BD. Esta abordagem acarreta, no entanto, alguns problemas no que diz respeito à manutenção de código. O facto de se utilizar uma `String` para a definição do comando SQL:

- Aumenta a probabilidade de erro humano do programador ao escrever o comando;
- Obriga a rever se um ou mais comandos se mantém válidos caso se alterem os nomes das colunas e/ou tabelas, se acrescentem novas colunas de valor obrigatório, se removam colunas, se definam novos relacionamentos, etc;
- A sua validação só pode ser realizada em *runtime*;
- Abre a possibilidade de realização de ataques por SQL Injection;

Por estes e outros motivos, têm vindo a ser desenvolvidas outras formas de acesso a BDs mais direccionadas para linguagens orientadas a objectos, das quais se destacam as bibliotecas de Object-Relational Mapping (O/RM). Uma biblioteca desta natureza permite:

- Mapear um modelo de objectos a um conjunto de tabelas numa BD, em que, normalmente, uma classe representa uma tabela e uma propriedade desta classe representa uma coluna;
- Executar comandos SQL sem recorrer a `String`s. Os comandos são gerados dinamicamente com base no mapeamento definido no ponto anterior;
- Definir uma camada que torna o acesso a uma BD transparente, dado que a maioria destas bibliotecas suporta mais que um SGBD sem necessitar de alterações no código da aplicação.

Actualmente existe uma grande variedade de bibliotecas de O/RM, tanto gratuitas, como comerciais, embora uma das mais conhecidas seja o Hibernate [5]. É uma biblioteca gratuita e *open-source* implementada em JAVA (existe um *port* para .NET, o NHibernate), suportada por uma comunidade com considerável dimensão, extensível e está integrada no Glassfish Server (através da implementação da Entity Manager API, da plataforma JAVA) e no NetBeans IDE. Para além disto, é possível encontrar extensões que acrescentam outras funcionalidades a esta biblioteca, como é o caso da HibernateSpatial [6], que fornece suporte para tipos de dados geográficos e funções espaciais para vários SGBDs – Oracle, SQL Server, PostgreSQL e MySQL. O seu modo de funcionamento resume-se a proceder à conversão automática entre os tipos de dados geográficos específicos de cada SGBD e objectos do tipo `Geometry` da biblioteca Java Topology Suite (JTS) [7], uma conhecida biblioteca para manipulação de dados geográficos (também existe um *port* para .NET desta biblioteca, a NetTopologySuite [8]). Desta forma, é possível continuar a manter o acesso transparente a uma BD para aplicações com requisitos desta natureza, dado que o código de uma aplicação apenas manipulará objectos da biblioteca JTS.

2.3.2.1. OpenStreetMap

O OpenStreetMap é um projecto *open-source,* composto por dezenas de aplicações, que pretende disponibilizar informação georreferenciada de forma livre, sem restrições de licenças ou encargos financeiros [9]. Esta informação pode ser obtida de várias formas, sendo as mais usuais as seguintes:

- A partir do sítio deste projecto sob forma de um ou mais ficheiros XML, estruturados de acordo com um *schema* desenvolvido pelo OpenStreetMap. A estes ficheiros dá-se o nome de OSM Files, dado a sua extensão ser ".osm" ou ".osm.bz2";

- Através de uma API, baseada no protocolo HTTP, para comunicação com os servidores OpenStreetMap que permite efectuar o pedido e a obtenção de informação georreferenciada armazenada nas bases de dados deste sistema. A figura 6 mostra o DER de alto nível de uma base de dados OpenStreetMap.

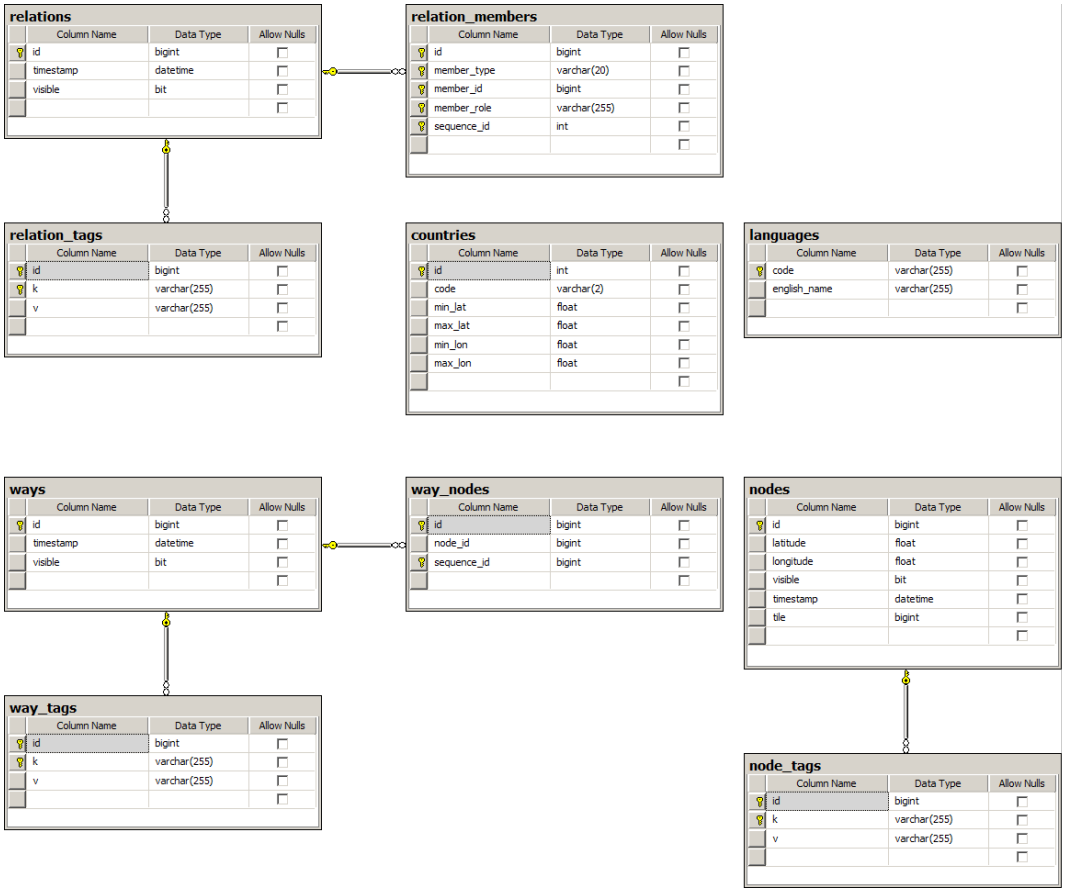

Figura 6 - DER de alto nível do OpenStreetMap

O modelo de dados do OpenStreetMap é definido pelos elementos Nó (*Node*), Caminho (*Way*), Etiqueta (*Tag*) e Relação (*Relation*).

Nós (*Nodes*)

Um Nó é o elemento básico do modelo do OpenStreetMap que contém uma latitude e longitude e representa, por isso, um determinado ponto no globo. Um Nó pode fazer parte de um Caminho, mas também poderão existir Nós isolados, sem ligação a qualquer outro Nó e que podem, por exemplo, corresponder a um telefone público, um bar, ou qualquer outro tipo de ponto de interesse. Os Nós isolados devem ter pelo menos uma Etiqueta. Por outro lado, os Nós que fazem parte de um Caminho normalmente não têm qualquer Etiqueta (apenas existem para definir o próprio Caminho), embora não se deva fazer desta afirmação uma regra. Por exemplo, um Caminho pode identificar um caminho de ferros e conter um Nó com uma Etiqueta que corresponda a uma estação de comboios.

Caminhos (Ways**)**

Um Caminho é uma interligação ordenada de pelo menos 2 Nós que descreve uma linha que pode representar uma rua, um caminho pedonal, um caminho de ferro, um rio, etc. Um Caminho é caracterizado por um conjunto uniforme de propriedades, por exemplo, a prioridade da via, a qualidade do pavimento, a velocidade permitida, etc. Este deve ser dividido em secções mais pequenas caso se verifiquem propriedades inconsistentes ao longo do percurso. Por exemplo, se uma rua tem uma zona de sentido único, então deve definir-se um Caminho específico para essa zona. É possível utilizar um Caminho para definir uma área, desde que a localização do primeiro Nó seja coincidente com a localização do último Nó.

Relações (*Relations*)

Uma relação é um grupo com zero ou mais elementos associados entre si, normalmente derivado à proximidade das suas localizações. Cada elemento do grupo pode desempenhar um papel no contexto da relação como, por exemplo, as relações "é proibido virar desta estrada para aquela estrada" ou "esta entrada dá acesso ao túnel do metro", definem o papel de proibição e permissão de passagem, respectivamente.

Etiquetas (*Tags*)

Uma Etiqueta consiste num par chave/valor que permite caracterizar determinado aspecto acerca de um Nó, Caminho ou Relação. Cada um destes elementos pode estar associado a um conjunto indeterminado de Etiquetas, mas esse conjunto não pode conter Etiquetas com chaves de igual valor. Já existem dezenas de pares chave/valor pré-definidos pelo OpenStreetMap que podem ser usados como Etiquetas, de forma a que a caracterização dos elementos do modelo seja feita o mais uniformemente possível pela comunidade. A tabela C dos anexos contém alguns exemplos de Etiquetas que se podem definir através de pares chave/valor.

2.3.3. Plataformas para desenvolvimento de aplicações *desktop*

Uma característica comum numa aplicação de dimensão empresarial é o facto de ser modular e extensível, capacidades que facilitam a adição de novas funcionalidades ou a alteração de funcionalidades já existentes. Pretende-se, por isso, desenvolver ou utilizar uma plataforma

que promova essa e outras capacidades, de modo a facilitar a sua evolução. A tabela 6 identifica algumas plataformas de desenvolvimento existentes organizadas pela linguagem de programação na qual foram implementadas. Todas elas têm como característica comum o facto de serem gratuitas, sendo que as plataformas Eclipse e NetBeans representam a base para o desenvolvimento de ferramentas de programação.

Plataforma	Linguagem	Licença	Observações
Smart Client Software Factory	C#	Ms-PL	Desenvolvida pela Microsoft. Não foram realizados mais desenvolvimentos desde Abril de 2008. Pensada para aplicações Windows Forms. A sua arquitectura torna o processo de testes complexo.
Prism	C#	Ms-PL	2ª geração de plataformas gratuitas desenvolvida pela Microsoft. Pensada para aplicações WPF e Silverlight, embora se possa adaptar para aplicações Windows Forms.
Eclipse RCP	Java	EPL	Desenvolvida pela IBM. Plataforma serve como base para um IDE. Utiliza o SWT como biblioteca de interface gráfica.
Netbeans Platform	Java	CDDL ou GPL	Desenvolvida pela Sun. Plataforma serve como base para um IDE. Utiliza o Swing como biblioteca de interface gráfica.

Tabela 6 - Plataformas para desenvolvimento de aplicações

2.3.4. Ferramentas SIG

O tabela 7 sintetiza algumas características das principais ferramentas SIG existentes no mercado. De realçar o facto de, hoje em dia, ser possível encontrar ferramentas deste tipo (algumas delas gratuitas, embora a grande maioria sob a licença GPL) desenvolvidas em várias linguagens, nomeadamente, C++, JAVA, C#, Python, entre outras.

Aplicação	Fabricante	Preço	Linguagem de programação	Observações
ArcGIS	ESRI	A partir de 3100 €	C++, VBA	Líder de mercado. É possível fazer integração com outras aplicações.
Geomedia	Intergraph	Geomedia Pro a partir de USD $7500	C++	É possível fazer integração com outras aplicações.
uDig	Refractions Research	Gratuito e Open Source (sob a licença LGPL)	Java	Baseado na plataforma Eclipse. É possível desenvolver módulos em Java, seguindo as convenções definidas pela própria plataforma Eclipse.
Quantum GIS	OSGeo	Gratuito e Open Source (sob a licença GPL)	C++, Python	É possível desenvolver módulos em Python.
OpenJUMP	Vivid Solutions	Gratuito e Open Source (sob a licença GPL)	Java	É possível desenvolver módulos em Java.
GvSIG	Conselleria de Infraestructuras y Transporte (Espanha)	Gratuito e Open Source (sob a licença GPL)	Java	Baseado no OpenJUMP. É possível desenvolver módulos em Java.
GRASS GIS	OSGeo	Gratuito e Open Source (sob a licença GPL)	C++, Python, TCL	Não suporta BD's Oracle.
SAGA GIS	SAGA User Group Association	Gratuito e Open Source (sob a licença LGPL)	C++	Não suporta BD's Oracle.
Ilwis	52° North	Gratuito e Open Source (sob a licença GPL)	C++	Não suporta BD's Oracle.
SharpMap	Grupo de voluntários	Gratuito e Open Source (sob a licença LGPL)	C#	Trata-se de um componente que pode ser integrado noutras aplicações.
MapWindow	Universidade de Idaho	Gratuito e Open Source (sob a licença MPL)	Versão 4: C++ Versão 6: C#	Não possui qualquer funcionalidade para manipular dados geográficos de uma BD.

Tabela 7 - Ferramentas SIG existentes no mercado

2.3.5. Dispositivos móveis existentes no mercado

Foram recolhidas as informações estatísticas disponibilizadas pela Canalys, uma empresa especializada em análise do mercado de tecnologias móveis, relativas ao terceiro trimestre de 2009 [10]. Essas informações permitem concluir o seguinte:

- As encomendas de dispositivos móveis aumentaram 3.9 %, em comparação com o período homólogo de 2008, com a venda de 41.4 milhões de unidades;
- A Nokia, RIM, Apple e HTC detêm em conjunto mais de 80 % do mercado dos dispositivos móveis;

- A região denominada pela Canalys por Ásia Pacífico cresceu cerca de 26,3 % enquanto que a região da Europa, Médio Oriente e África decresceu 5,8 %;
- A proporção de *smart phones* com *touchscreen* atinge os 45 %;
- Mais de 80 % dos *smart phones* fabricados actualmente vêm com GPS integrado e mais de 75 % com Wi-Fi (suporte para redes sem fios).

A tabela 8 mostra a comparação do número de encomendas de dispositivos móveis efectuadas no terceiro trimestre de 2009 com o terceiro trimestre de 2008, relativamente a cada região do globo.

Região	**3º Trimestre 2009**		**3º Trimestre 2008**		**Crescimento homólogo**
	Encomendas	**% mercado**	**Encomendas**	**% mercado**	
Ásia Pacífico	13226770	31,95	10470990	26,28	26,32
Europa, Médio Oriente e África	13950770	33,7	14808640	37,16	-5,79
América Latina	2051930	4,96	2951760	7,41	-30,48
América do Norte	12164780	29,39	11618710	29,16	4,7
Total	**41394250**	**100**	**39850100**	**100**	**3,87**

Tabela 8 - Encomendas de dispositivos móveis por região

A figura 7 representa graficamente os dados da tabela 8 e, como é possível observar, apenas as regiões da Ásia Pacífico e América do Norte cresceram no terceiro trimestre de 2009, enquanto que as regiões da Europa, Médio Oriente, África e América Latina decresceram em igual período, destacando-se, no entanto, a América Latina com uma descida de 30,48 %.

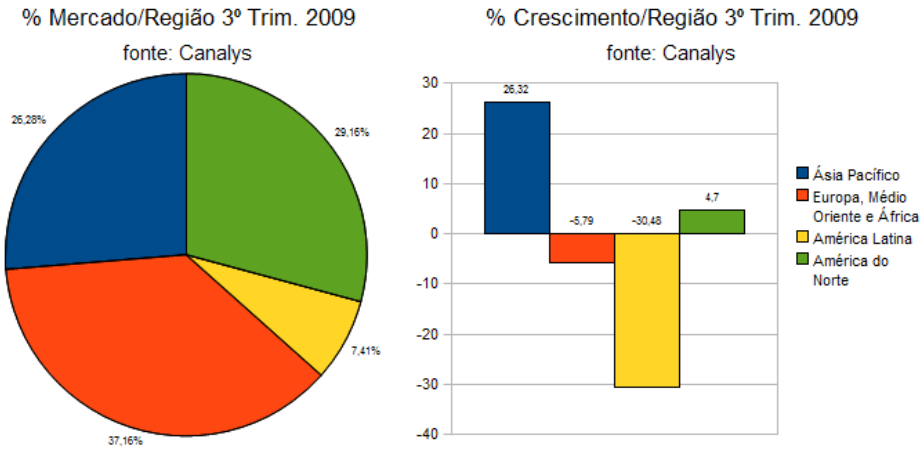

Figura 7 - Percentagem de mercado de dispositivos móveis por região

De seguida, é apresentada a tabela 9 que faz a comparação de encomendas realizadas no terceiro trimestre de 2009 por fabricante de dispositivos móveis.

Fabricante	3º Trimestre 2009		3º Trimestre 2008		Crescimento homólogo
	Encomendas	% mercado	Encomendas	% mercado	
Nokia	16413420	39,65	15485690	38,86	5,99
RIM	8521280	20,59	6051730	15,19	40,81
Apple	7632670	17,79	6899010	17,31	6,72
HTC	2179690	5,27	2308210	5,79	-5,56
Fujitsu	1394410	3,37	1093870	2,74	27,47
Outros	5522510	13,34	8011590	20,1	-31,07
Total	**41394250**	**100**	**38850100**	**100**	**3,87**

Tabela 9: Encomendas de dispositivos móveis por fabricante

A figura 8 revela claramente que a Nokia é líder de mercado no terceiro trimestre de 2009 com quase 40 %, condição que também se verificava em 2008. Por outro lado, a RIM apresentou um crescimento de mais de 40 %, ultrapassando já os 20% de mercado.

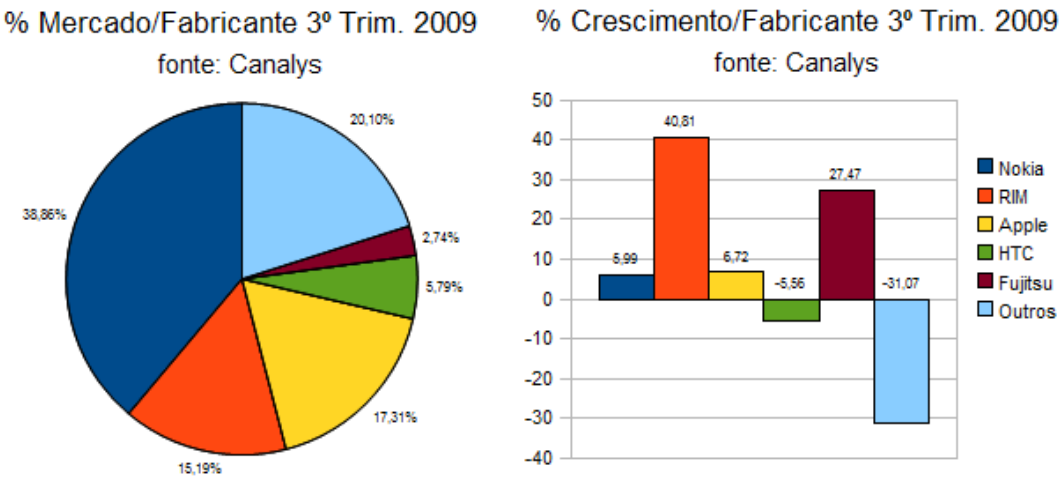

Figura 8 - Percentagem de mercado e crescimento por fabricante

A tabela 10 faz o resumo das encomendas feitas relativamente a sistemas operativos para dispositivos móveis, sustentando os dados das restantes tabelas e figuras apresentadas anteriormente. De destacar a entrada da Google com o seu sistema operativo Android, para o qual não é possível obter o crescimento homólogo, por este ainda não existir no terceiro trimestre de 2008.

Sistema Operativo	3º Trimestre 2009		3º Trimestre 2008		Crescimento homólogo
	Encomendas	% mercado	Encomendas	% mercado	
Symbian	19107490	46,16	18583060	46,63	2,82
RIM	8521280	20,59	6051730	15,19	40,81
Apple	7362670	17,79	6899010	17,31	6,72
Microsoft	3631630	8,77	5425470	13,61	-33,06
Google (Android)	1455140	3,52	N/D	N/D	N/D
Outros	1316040	3,18	2890830	7,25	-54,48
Total	**41394250**	**100**	**39850100**	**100**	**3,87**

Tabela 10 - Encomendas de dispositivos móveis por sistema operativo

Como se pode observar na figura 9, o sistema operativo Symbian ocupa mais de 46 % mercado, muito devido ao facto de ser este o sistema operativo usado pela Nokia nos seus dispositivos.

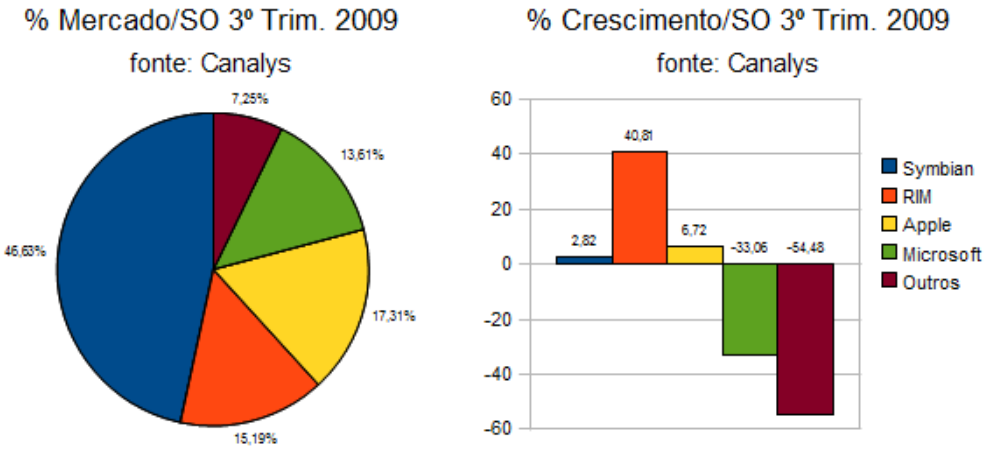

Figura 9 - Percentagem de mercado e crescimento por sistema operativo

2.3.6. Dispositivos móveis para fins empresariais

Embora o mercado de dispositivos móveis para o público em geral seja já tecnologicamente evoluído, existem outros dispositivos para fins empresariais, designados por computadores de campo (*field computers* ou *rugged computers*), que apresentam características que normalmente são difíceis de encontrar nos modelos das marcas referidas na secção anterior. Estas características estão relacionadas com a resistência dos materiais de fabrico, a precisão dos componentes, a ergonomia do dispositivo, entre outras. A tabela 11 enumera alguns fabricantes que produzem dispositivos de gama empresarial.

Fabricante	SO's suportados	Endereço
Juniper Systems	Windows Mobile	http://www.junipersys.com/
Trimble	Windows Mobile	http://www.trimble.com/index.aspx
General Dynamics Itronix	Windows XP, Windows Vista, Windows 7, Windows XP Tablet Edition	http://www.gd-itronix.com/
Raven	Windows XP	http://www.ravenprecision.com/
Topcon	Windows CE	http://www.topcon-positioning.eu/index.asp
Xplore Technologies	Windows 7, Windows XP Tablet Edition	http://www.xploretech.com/
Intermec	Windows Mobile	http://www.intermec.com/products/
Psion Teklogix	Windows Mobile, Windows CE	http://www.psionteklogix.com/
Motorola	Windows Mobile	http://www.motorola.com/Business/US-EN/Enterprise+Mobility
DAP Technologies	Windows CE, Windows Mobile	http://www.daptech.com/

Tabela 11 - Fabricantes de dispositivos móveis para fins empresariais

Como é possível observar pelos dados da tabela 11, todos os fabricantes dão suporte apenas a sistemas operativos Windows, sendo que a maioria suportam apenas o Windows Mobile (à excepção dos produtos da Itronix e da Xplore Technologies que se baseiam em Notebooks e/ou Tablet PCs). Para obter mais informações para além das características técnicas dos diversos dispositivos móveis apresentados nos *sites* de cada fabricante (por exemplo, preços), é necessário entrar em contacto com o próprio fabricante ou com os seus revendedores.

2.3.7. Linguagens de programação e bibliotecas para aplicações móveis

De modo a identificar qual o *software* a usar para desenvolver uma aplicação móvel é necessário ter em conta os sistemas operativos dos dispositivos para os quais se pretende disponibilizar a aplicação. Existem disponíveis um conjunto de linguagens e SDKs que são normalmente diferentes para cada sistema operativo. As linguagens mais usadas para a implementação de aplicações móveis são: JAVA (para dispositivos com a máquina virtual Java instalada), C++ (para sistemas operativos Symbian, Windows CE, Windows Mobile, entre outros), .NET (para dispositivos com a .NET Compact Framework instalada) e Objective-C (para os IPhones da Apple).

2.3.8. SGBDs para aplicações móveis

A tabela seguinte compara algumas das principais características dos SGBDs mais conhecidos para dispositivos móveis.

Nome	Multi-Plataforma	Código Aberto?	Preço	Suporte para SIG?
SQLite	✓	✓	Gratuito	✓
Sybase SQL Anywhere	✓	✗	103,20 €	✗
IBM DB2 Everyplace	✓	✗	USD $72,5	✗
SQL Server Compact	✗	✗	Gratuito	✗
Oracle Lite 10g	✓	✗	USD $60,0	✗

Tabela 12 - Comparação entre SGBDs para dispositivos móveis

Como se pode verificar pelos dados da tabela 12, o SQLite é o SGBD mais adequado para o desenvolvimento de uma aplicação móvel, dado ser gratuito, multi-plataforma e ter suporte para dados geográficos, através da utilização de uma extensão denominada Spatialite que acrescenta funcionalidades SIG ao motor (tipos de dados geográficos e funções espaciais).

2.3.9. Ferramentas SIG para dispositivos móveis

Um dos aspectos de interesse para a aplicação móvel a implementar diz respeito à visualização de dados geográficos relativos às estruturas a inspeccionar. A tabela 13 apresenta um levantamento de ferramentas SIG actualmente existentes com suporte para plataformas móveis. A partir desta tabela pode verificar-se que:

- Só foi encontrada uma ferramenta SIG móvel gratuita – o GvSIG Mobile, implementado em JAVA;
- A maioria destas ferramentas têm como requisito o Windows Mobile;
- Todos estas ferramentas disponibilizam, no geral, as mesmas funcionalidades:
  - Leitura, visualização, edição e pesquisa de geometrias;
  - Navegação de mapas;
  - Integração com o receptor GPS.

Nome	Fabricante	Requisitos	Observações
ArcGIS Mobile, ArcPad	ESRI	Windows Mobile	Líder de mercado; Permite a visualização e navegação de mapas; Recolha, edição e actualização de dados geográficos; Pesquisar e gerir dados geográficos; Implantação de sistemas móveis SIG sem a necessidade de desenvolvimentos; Integração com o receptor GPS;
GvSIG Mobile	Conselleria d'Infraestructures i Transports	Qualquer dispositivo que suporte o J2ME Personal Profile 320 MHz CPU 64 MB RAM	Gratuito (sob a licença GPL); Implementado em Java; Permite a visualização e navegação de mapas; Recolha, edição e actualização de dados geográficos; Pesquisar e gerir dados geográficos; Integração com o receptor GPS;
PocketGIS	Pocket Systems	Windows CE, Windows Mobile	500 Libras (UK) por licença; Permite a visualização e navegação de mapas; Recolha, edição e actualização de dados geográficos; Pesquisar e gerir dados geográficos; Integração com o receptor GPS;
MapXtend	Pitney Bowes Business Insight	Qualquer dispositivo com J2ME	Implementado em Java; Permite a visualização e navegação de mapas; Recolha, edição e actualização de dados geográficos; Pesquisar e gerir dados geográficos; Integração com o receptor GPS;
HGIS	Star Pal	Windows Mobile Resolução mínima 240x240	Permite a visualização e navegação de mapas; Recolha, edição e actualização de dados geográficos; Pesquisar e gerir dados geográficos; Integração com o receptor GPS;
MapFrame	GE Energy	Qualquer dispositivo com SO Windows instalado	Permite a visualização e navegação de mapas; Recolha, edição e actualização de dados geográficos; Pesquisar e gerir dados geográficos; Integração com o receptor GPS;

Tabela 13 - SIG's para dispositivos móveis

2.3.10. Servidores aplicacionais

Um servidor aplicacional é um sistema que permite o desenvolvimento de aplicações distribuídas. Funciona como um contentor onde se podem hospedar essas aplicações, fornecendo-lhes um conjunto de serviços que são quase sempre necessários para a sua execução, como por exemplo, gestão de ligações e transacções à BD, gestão e injecção de recursos, entre outros [11]. Existem servidores aplicacionais implementados nas mais diversas linguagens, embora os mais conhecidos sejam baseados na plataforma Java 2, Enterprise

Edition (J2EE), sendo que a grande maioria tem licença comercial. Destaca-se, no entanto, o Glassfish Server por ser gratuito e *open-source* e por ser possível desenvolver, instalar, depurar e testar o código da aplicação directamente a partir dos principais IDEs para programação em JAVA, nomeadamente, o Eclipse, o NetBeans IDE, etc [12].

2.3.11. Conclusões

Foram apresentadas algumas das muitas tecnologias actualmente existentes para a implementação de uma aplicação *desktop* e de uma aplicação móvel. A investigação efectuada incidiu principalmente em linguagens de programação, SGBDs, plataformas de desenvolvimento, dispositivos móveis e ferramentas SIG. Os dados recolhidos permitem retirar as seguintes conclusões:

- Para aplicações de média e grande dimensão com requisitos de gestão de informação geográfica, o SGBD recomendado é o Oracle, mesmo tendo em conta que será necessário investir mais de USD $65.000 só pela sua aquisição. Para aplicações de pequena dimensão, o SGBD recomendado é o PostgreSQL, dado ser gratuito, estável e rico em funcionalidades geográficas;

- Java é a linguagem de programação recomendada para a implementação deste género de aplicações, pois:

    - é a linguagem utilizada no desenvolvimento do servidor aplicacional Glassfish;

    - é a linguagem escolhida pelo Oracle e pelo PostgreSQL para o desenvolvimento das suas bibliotecas para interacção com o SGBD, incluindo as que estão directamente relacionadas com funcionalidades espaciais.

    - Existem centenas de outras bibliotecas, ferramentas, plataformas e aplicações gratuitas implementadas em Java para os mais diversos fins e que, por isso, permitem apetrechar uma equipa de desenvolvimento com soluções *out-of-the-box*, reduzindo assim custos e tempo;

- NetBeans é a plataforma de desenvolvimento recomendada por estar implementada em Java e por utilizar o Swing. Swing é uma biblioteca para desenvolvimento de

interfaces gráficas que vem incluída no JDK e já existem dezenas de bibliotecas gratuitas que complementam as suas funcionalidades, facto que não se verifica em comparação com a biblioteca SWT;

- A Nokia é a marca de dispositivos móveis que mais vende a nível global. Este facto deve ser considerado como meramente informativo, pois o público-alvo do sistema móvel a desenvolver não é o público em geral. É uma aplicação que será utilizada por especialistas em obras marítimas, cujo perfil tecnológico se prevê ser mais elevado do que o da restante população, em média. Para além disto, os dispositivos da Nokia têm um sistema operativo – o Symbian – que apenas suporta C++ (ou Java, caso possua uma máquina virtual instalada) como linguagem de programação;

- Existem vários fabricantes especializados na produção de dispositivos móveis para fins empresariais que proporcionam outras características que não se encontram no mercado de dispositivos para o público em geral. A larga maioria destes dispositivos traz instalado o sistema operativo Windows Mobile, tal como é indicado na tabela 11;

- De forma a facilitar o processo de escolha de um dispositivo móvel, devem ser definidos critérios de selecção de acordo com o modo de utilização do dispositivo e o contexto em que será utilizado. Poderá ser necessário ter em consideração:

    - O preço;
    - A resistência / qualidade dos materiais de fabrico;
    - A ergonomia do dispositivo;
    - A precisão dos vários componentes constituintes, como por exemplo, o receptor GPS;
    - O tamanho do ecrã;
    - Se o ecrã deve ser sensível ao toque (*touchscreen*);
    - A resolução da câmara;

- A duração da bateria;

- O SQLite é o SGBD recomendado para aplicações móveis, por ser gratuito, multi-plataforma e por suportar tipos de dados geográficos. Caso não sejam requeridas funcionalidades SIG, o SQL Server Compact apresenta-se como uma boa alternativa;

- O OpenStreetMap implementa um modelo para estruturação de dados que pode ser utilizado como referência para a criação de repositórios de geometrias.

3. Arquitectura

O sistema desenvolvido é composto fisicamente pelos elementos que se encontram representados na figura 10. Para a sua implementação, foi necessário desenvolver três aplicações: uma aplicação *desktop*, uma aplicação móvel e uma aplicação servidor.

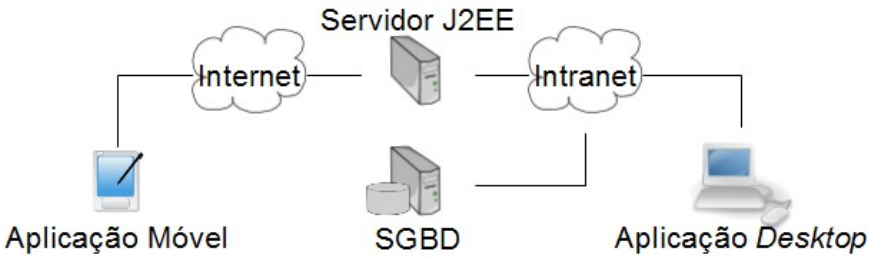

Figura 10 - Visão geral do sistema

Tal como foi referido anteriormente, os objectivos principais a atingir com este sistema são:

- Definir as perguntas e respostas possíveis de uma ou mais fichas de inspecção. Esta tarefa está a cargo do utilizador que gere a aplicação *desktop*, que também pode introduzir e caracterizar obras marítimas, assim como assinalar uma ou mais obras para inspecção. Para além disto, deve ser possível consultar as fichas de inspecção preenchidas;

- Disponibilizar remotamente os dados das fichas de inspecção definidas no ponto anterior, incluindo os dados relativos às obras marítimas a inspeccionar. Esta tarefa é assumida pelo servidor J2EE, que também permite o envio dos dados das fichas de inspecção preenchidas;

- Preencher uma ou mais fichas de inspecção. Esta tarefa é levada a cabo pelo utilizador que gere a aplicação móvel (o inspector), que, após o preenchimento dessas fichas, tem a possibilidade de as enviar para servidor J2EE.

A metodologia aplicada para o desenvolvimento das três aplicações é orientada segundo o Domain-Driven Design (DDD), que define que uma aplicação deve ser estruturada em camadas, separadas pelas responsabilidades que cada uma assume [13]. A figura 11 ilustra como se pode organizar uma aplicação em cinco camadas:

- **Modelo de domínio** – Representa todos os conceitos e regras de negócio e o estado actual em que este se encontra. Esta é a camada principal de uma aplicação;
- **Acesso a dados** – Responsável por gerir a persistência dos objectos do modelo de domínio. Normalmente, são armazenados numa BD;
- **Lógica aplicacional** – Define as acções que o utilizador pode realizar com a aplicação e delega o processamento para os objectos do modelo de domínio ou para outras camadas da aplicação;
- **Interface com o utilizador** (ou **Apresentação**) – Responsável por apresentar informação ao utilizador e interpretar os seus comandos;
- **Infra-estrutura** – Disponibiliza funcionalidades genéricas que dão suporte às restantes camadas da aplicação.

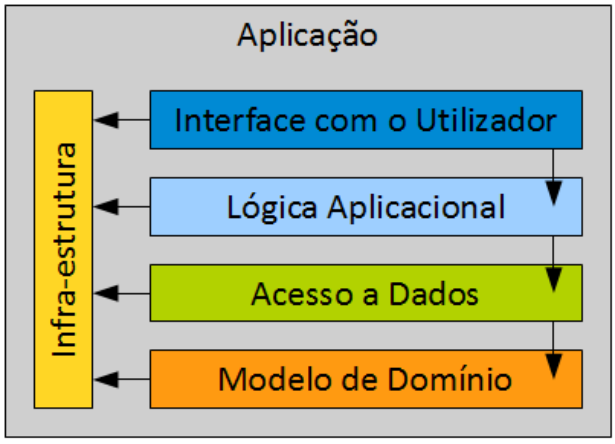

Figura 11 - Estruturação de uma aplicação por camadas

Esta arquitectura implica, no entanto, a realização de desenvolvimentos que, por si só, não dizem directamente respeito aos objectivos propostos por este projecto. Servem, fundamentalmente, como suporte aos modelos e módulos relativos ao tema da monitorização de obras marítimas.

As próximas secções descrevem em detalhe a arquitectura de cada uma das aplicações implementadas.

3.1. Arquitectura da aplicação desktop

A arquitectura da aplicação *desktop* tem como base a plataforma NetBeans, que possui um conjunto de bibliotecas e APIs que permitem desenvolver uma aplicação JAVA por módulos. Os primeiros módulos implementados têm como objectivo colmatar algumas limitações da própria plataforma, respeitantes predominantemente à gestão de dados, de utilizadores e grupos e que darão suporte aos módulos de negócio a implementar. A próxima imagem dá uma visão geral dos módulos utilizados e implementados para a aplicação *desktop*.

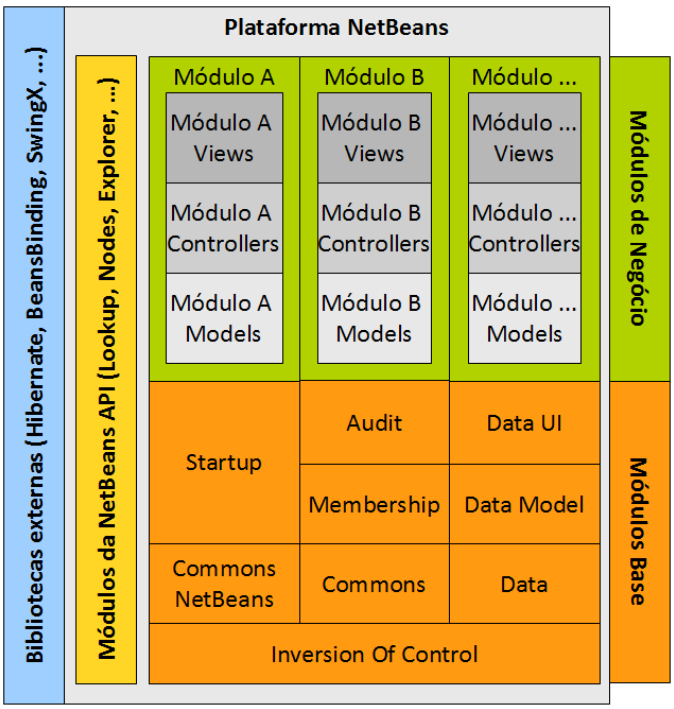

Figura 12 - Visão geral dos módulos da aplicação desktop

As próximas secções descrevem pormenorizadamente a implementação de alguns dos principais módulos da aplicação. Estes podem ser divididos em duas categorias:

- **Módulos Base** – Implementam parcial ou totalmente algumas das camadas ilustradas na figura 11 – Acesso a dados, Interface com o Utilizador e Infra-estrutura. É composta pelos seguintes módulos:
  - **Inversion of Control** – Implementação do padrão de desenho Inversion of Control Container (ou Dependency Injection);
  - **Commons** e **Commons NetBeans** – Composto por um conjunto de interfaces e classes utilitárias para auxiliar na resolução de problemas de programação comuns. O módulo Commons NetBeans, tal como o nome indica, contém código que apenas pode ser utilizado na implementação de módulos NetBeans;
  - **Data Module** – Implementação da camada de Acesso a Dados;
  - **Data UI** – Implementação da camada de Interface com o Utilizador;
  - **Membership** – Módulo de gestão de utilizadores;
  - **Startup** – Gere o início da execução da aplicação, momento em que é efectuada a autenticação do utilizador e a inicialização da camada de Acesso a Dados.
- **Módulos de Negócio** – Implementam em concreto os requisitos da aplicação, aqueles que dizem respeito à monitorização de obras marítimas. É composta pelos seguintes módulos:
  - **Geometry Model** – Modelo que define um repositório de geometrias baseado no OpenStreetMap;
  - **Inspections Model** – Modelo que define, de forma genérica, a estrutura de uma ficha de inspecção;
  - **Structures Model** – Definição de um modelo de gestão de estruturas marítimas. Tem uma dependência para o módulo Inspections Model, dado estas estruturas representarem os objectos que serão submetidos a inspecções.

3.1.1. Inversion Of Control

Este módulo implementa o padrão de desenho Inversion Of Control Container (embora também seja designado por Dependency Injection), recorrendo à biblioteca PicoContainer, e é normalmente encontrado em aplicações de arquitectura modular. De uma forma geral, este padrão define um contentor onde se podem registar serviços, normalmente definidos através de uma interface (ou classes abstractas), e as suas implementações, normalmente definidas por classes instanciáveis que implementam a interface do serviço. O restante código que utiliza este padrão pode obter uma instância de uma implementação, pesquisando no contentor pela interface. A instanciação do serviço pode estar sujeita à resolução de dependências por parte do contentor, um processo recursivo que é executado caso a implementação tenha parâmetros definidos no construtor. Se os tipos dos parâmetros representarem interfaces de serviços contidos no contentor, então este procede à instanciação desses serviços, injectando-os no construtor (daí o nome Dependency Injection) [14]. As vantagens que esta capacidade traz têm um enorme peso no desenvolvimento e manutenção de código e na realização de testes de unidade:

- A lógica da aplicação utiliza predominantemente interfaces, o que minimiza drasticamente o impacto da mudança de uma implementação, facilitando por isso a tarefa de manutenção de código;
- Reduz as dependências de código;
- A injecção torna o código extensível, pois basta acrescentar um ou mais parâmetros num construtor de uma implementação para se ter obter uma ou mais instâncias de serviços.
- Permite isolar determinadas partes de um sistema para fins de implementação de testes de unidade. Dado que o código da aplicação manipula interfaces, torna-se possível substituir as suas implementações por outras mais simples (ou mesmo implementações "vazias"), facilitando assim a criação de testes, dado que estes normalmente apenas avaliam uma parte muito específica do código.

Como foi referido no início desta secção, o padrão de desenho Inversion of Control é normalmente encontrado em plataformas de arquitectura modular e o NetBeans não foge a esta regra, embora apresente algumas diferenças a este padrão. Existe um módulo pertencente à NetBeans API, denominado de Lookup, que contém uma classe com o mesmo nome, cujo funcionamento se aproxima mais daquele que é definido pelo padrão de desenho Service Locator. Este padrão distingue-se do Inversion Of Control, por não ser possível registar serviços, havendo apenas a funcionalidade de pesquisa e instanciação. Para a Lookup API, esse registo só pode ser realizado em *compile time*, recorrendo aos mecanismos definidos pela ServiceLoader API do J2SE v1.6. Isto significa que não é possível efectuar o registo de serviços dinamicamente (em *runtime*) e que as implementações têm de definir um construtor por omissão (sem parâmetros), eliminando, por isso, qualquer possibilidade de injecção de dependências de forma automática. Existe sempre a hipótese de realizar a injecção manualmente, mas isso implicaria que todas as implementações passassem a estar dependentes do próprio contentor, o que contrapõe o objectivo principal deste padrão (que é precisamente reduzir as dependências de código).

O módulo Inversion of Control integra as funcionalidades da Lookup API com a biblioteca PicoContainer, acrescentando à instância de `Lookup` predefinido da plataforma NetBeans (obtido através do método `Lookup.getDefault()`) a capacidade de permitir efectuar o registo de serviços em *runtime* e de detectar e resolver dependências. O novo `Lookup` criado encontra-se representado no diagrama de classes da figura 13 com nome de `GlobalLookup`. De modo a integrar este módulo na plataforma NetBeans foi necessário:

1. Definir a classe como sendo descendente de `Lookup`;

2. Publicar métodos do contentor para o registo de serviços. O contentor é implementado recorrendo à biblioteca PicoContainer;

3. Implementar e registar um Monitor no contentor, um conceito definido pela biblioteca PicoContainer que permite o controlo fino sobre o processo de resolução de dependências. Sempre que não é possível resolver uma dependência automaticamente, o contentor, em último recurso, delega essa responsabilidade para um Monitor, o que lhe permite efectuar a resolução de forma não automática. O Monitor implementado

tenta resolver essa dependência pesquisando no `Lookup` predefinido uma instância do serviço a resolver.

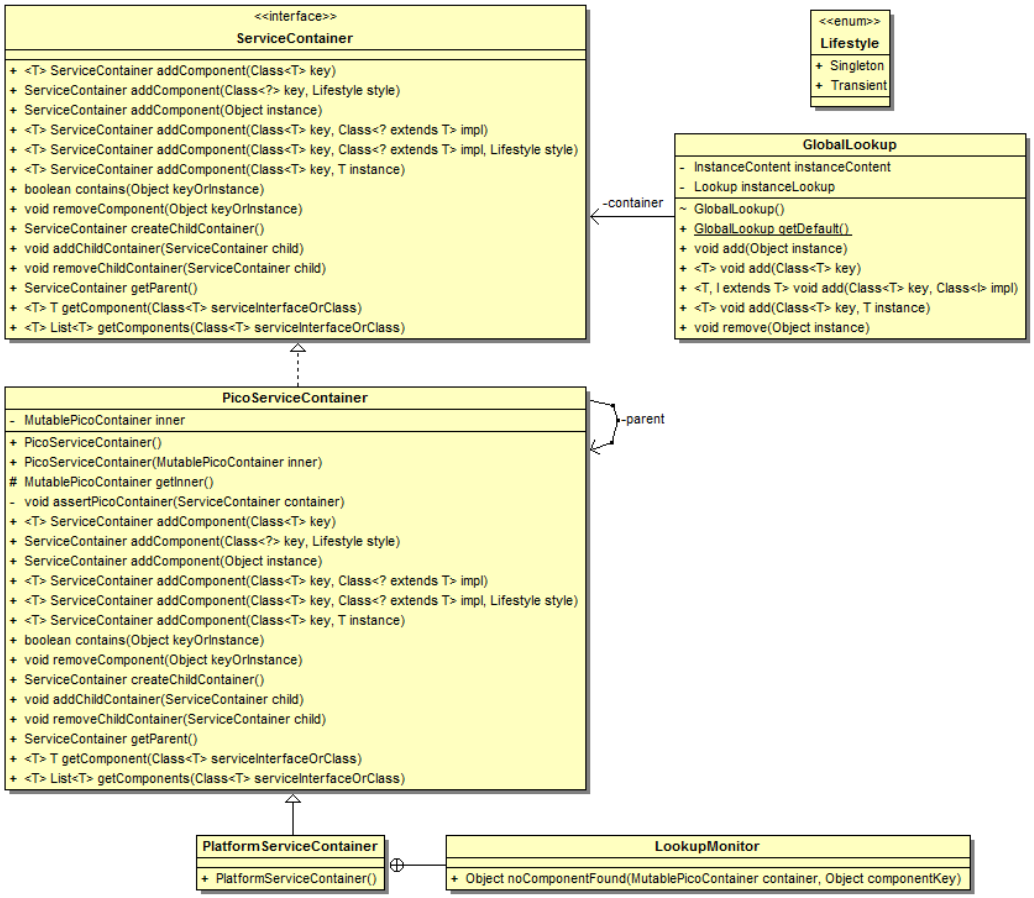

Figura 13 - Diagrama de classes do módulo Inversion of Control

O registo de serviços em *runtime* é realizado numa fase inicial do ciclo de vida de um módulo NetBeans, definido pela classe `ModuleInstall` da Module System API. Esta classe define cinco métodos que representam as fases existentes do ciclo:

1. **validate** – Invocado antes do módulo ser inicializado ou carregado. Esta fase é dedicada à análise de aspectos que possam impedir o carregamento do módulo, como por exemplo, verificar a existência e validade de um ficheiro de licenças. Caso se conclua que o módulo não deve ser carregado, deve ser lançada uma excepção do tipo `IllegalStateException`;

2. **restored** – Representa o momento da inicialização do módulo. São executadas todas as tarefas necessárias para o normal funcionamento do módulo. É nesta fase que se procede ao registo de serviços definidos pelo módulo;

3. **uninstalled** – Invocado quando o módulo é removido da aplicação;

4. **closing** – Invocado no momento antes da aplicação ser fechada. Permite testar se o módulo está num estado que permita ser terminado. Só é possível terminar a aplicação quando todos os módulos concluírem a execução deste método;

5. **close** – Invocado quando a aplicação está pronta para ser fechada.

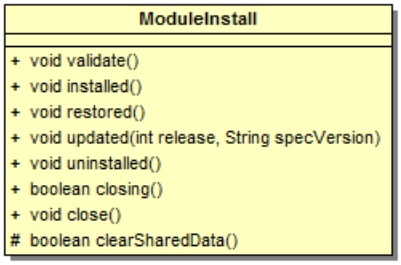

Figura 14 - A classe ModuleInstall

3.1.2. Data

O módulo de dados adiciona à aplicação a capacidade de gerir dados provenientes de uma base de dados. Integra no sistema, de forma transparente, o Hibernate, uma biblioteca de Object-Relational Mapping (O/RM) que faz o mapeamento entre um modelo relacional de uma BD e um modelo de objectos de uma aplicação. O mapeamento é definido recorrendo a anotações da Java Persistence API (JPA). Este módulo pode ser decomposto em três partes, de acordo com o problema que cada uma pretende resolver:

- **Repository** (Repositório) – É um padrão de desenho que implementa o conceito de colecção de objectos, sendo que essa colecção se encontra fisicamente armazenada numa base de dados [15]. O acesso aos dados é realizado por um repositório, que efectivamente serve como intermediário entre o Modelo de Domínio e a camada de Acesso a Dados (suportada pelo Hibernate), fazendo com que a utilização dessa camada seja feita de forma transparente. Isto permite que no futuro seja possível substituir o Hibernate por outra biblioteca de O/RM, caso seja necessário, sem que

essa mudança tenha impacto no código da aplicação.

Uma regra de ouro para a implementação deste padrão de desenho é que um Repositório não pode conter qualquer código relacionado com gestão de transacções, sendo esta responsabilidade delegada para a Unidade de Trabalho;

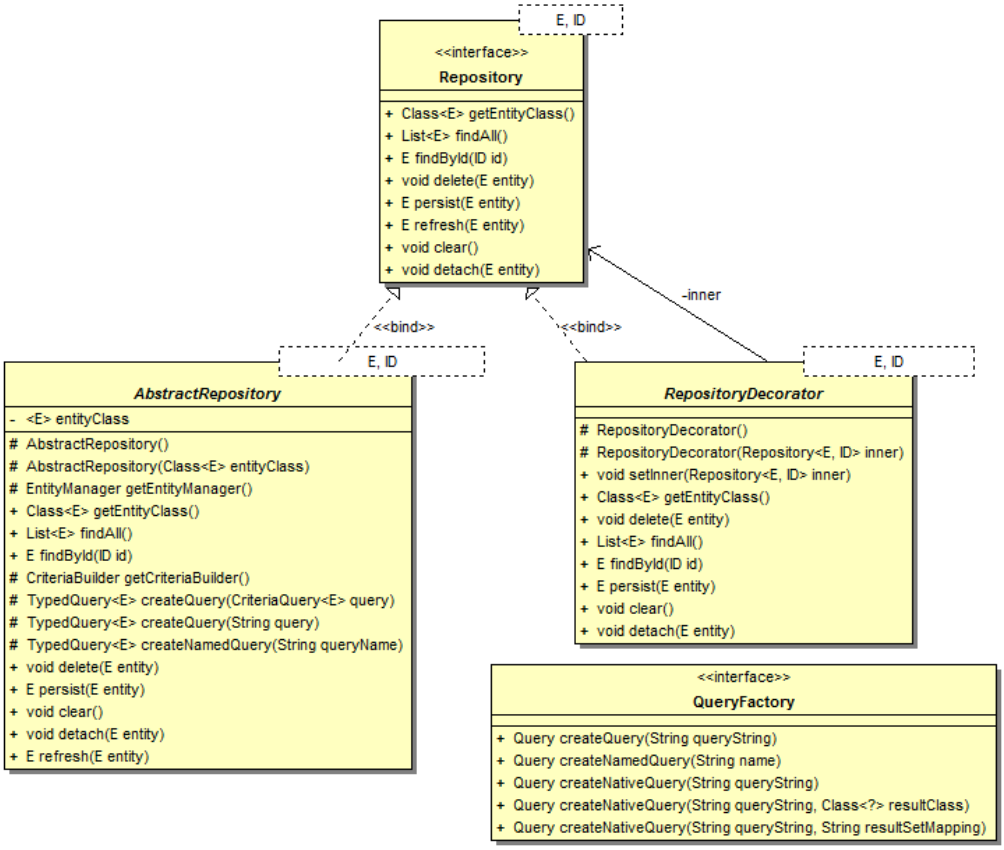

Figura 15: Diagrama de classes do padrão de desenho Repository

- **Unit of Work** (Unidade de Trabalho) – Padrão de desenho que tem como responsabilidade principal gerir os recursos necessários ao estabelecimento de sessões e transacções à base de dados [15]. O acesso aos Repositórios só pode ser feito após ter sido dado início a uma Unidade de Trabalho, caso contrário é lançada uma excepção. Após ter sido concluído o diálogo entre a aplicação e o SGBD, o programador tem a responsabilidade de fechar a Unidade de Trabalho, de forma a libertar todos os recursos usados até ao momento, nomeadamente, à ligação à BD.

  Outra funcionalidade implementada diz respeito ao registo das classes que representam entidades, sendo este registo imposto pela biblioteca Hibernate – todos os módulos podem registar *listeners* (classes que implementam a interface

`DataAccessInitializationListener`) que são informados, através do lançamento de eventos, sobre o estado de execução da inicialização deste módulo. Os eventos são disparados em três momentos:

- Antes de dar início à configuração – neste evento cada módulo tem a possibilidade de registar as entidades e definir outros parâmetros respeitantes à inicialização da camada de Acesso a Dados;
- Após ter sido realizada a configuração;
- Após ter sido inicializado com sucesso – neste evento cada módulo tem a possibilidade de realizar processamento adicional de modo a garantir o seu correcto funcionamento. Por exemplo, é neste momento que são registados no contentor de serviços, descrito na secção 3.1.1.

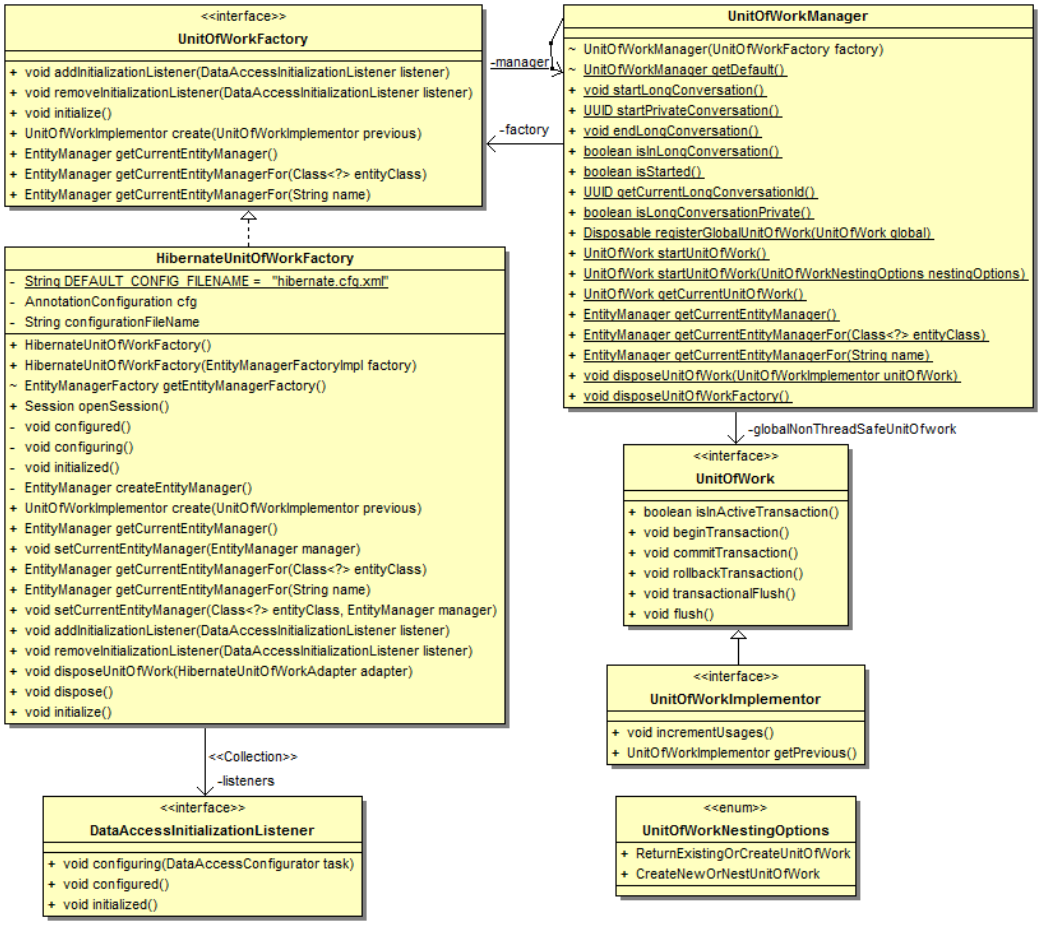

Figura 16 - Diagrama de classes do padrão de desenho Unit of Work

- **Entity** (Entidade) – Fornece a base que todas as classes mapeadas devem herdar. As funcionalidades que disponibiliza são as seguintes:
  - Permite determinar se um objecto é ou não transiente (se foi ou não armazenado na base de dados);
  - Contém uma implementação do método `equals(Object)` que define, por omissão, que um objecto é idêntico a outro se for do mesmo tipo e se o identificador (a propriedade que representa a chave primária) tiver o mesmo valor;
  - Disponibiliza métodos que possibilitam a observação de alterações em qualquer um dos atributos do objecto;
  - Fornece uma solução para o problema relacionado com a inicialização de objectos ou colecções *lazy* de uma entidade quando a sessão de acesso à base de dados se encontra fechada. A solução envolve a aplicação de técnicas de programação orientada a aspectos, recorrendo à biblioteca AspectJ [16]. O aspecto encontra-se representado no seguinte diagrama de classes pelo nome de `LazyInitializationAspect` e é aplicado a todos os atributos de uma entidade que representem relacionamentos. Um relacionamento é identificado se ao atributo em causa tiver sido aplicado uma das seguintes anotações JPA: `@OneToMany`, `@ManyToOne`, `@ManyToMany` ou `@OneToOne`. O código do aspecto é injectado após a compilação, através de técnicas de instrumentação de *bytecode* (processo executado automaticamente pelo AspectJ), de modo a interceptar a invocação a um *getter* que possua uma das anotações JPA referidas anteriormente. Assim, é possível detectar com antecedência se um objecto ou colecção ainda não foi inicializado e, caso isso se verifique, o aspecto estabelece novamente a sessão Hibernate, de modo a ser possível obter os dados da BD, permitindo assim realizar a inicialização correcta do atributo em causa. Desta forma, não só é possível tratar o lançamento de excepções do tipo `LazyInitializationException` (pertencente à biblioteca Hibernate) automaticamente e sem qualquer inclusão de código de gestão de dados nos

modelos de domínio, como também se adquire a capacidade "mágica" de obtenção de dados provenientes de uma BD invocando apenas um *getter* de um objecto de uma entidade.

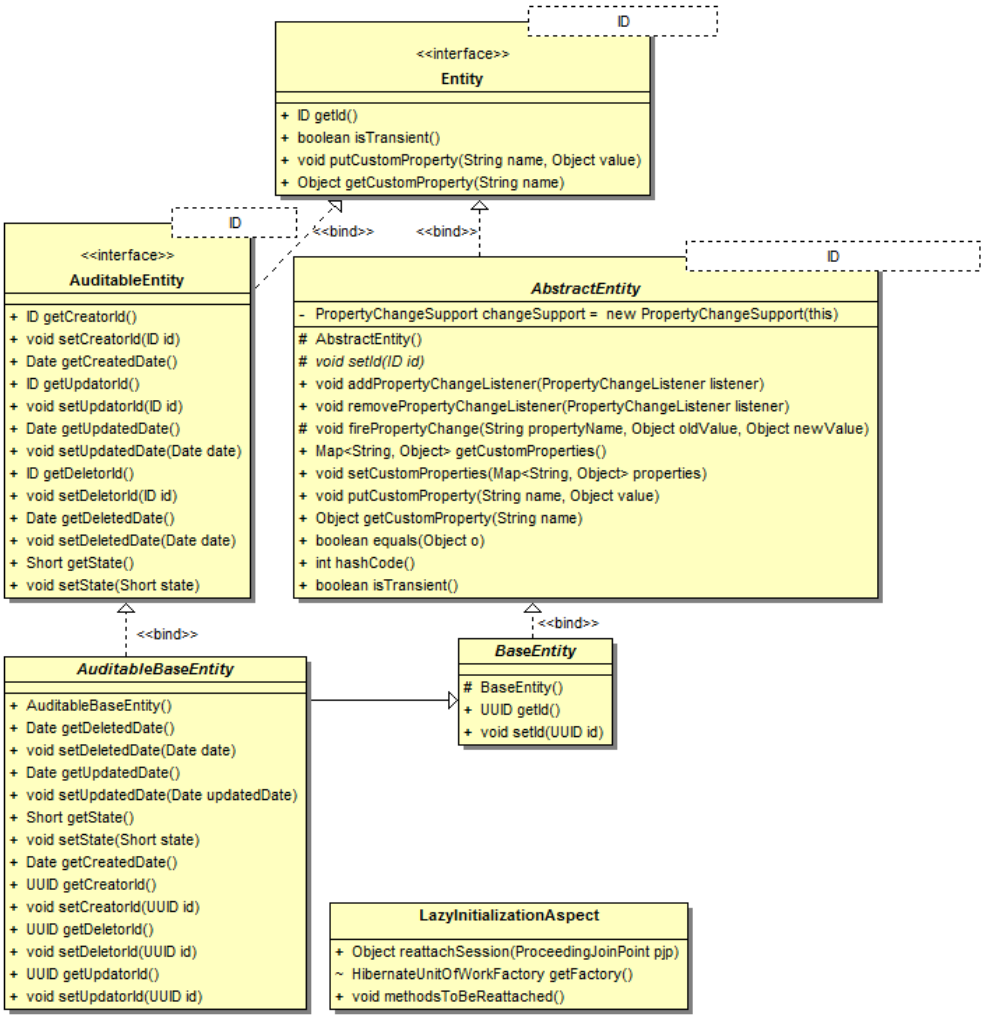

Figura 17 - Diagrama de classes do modelo de entidades

Note-se que o diagrama de classes apresentado anteriormente mostra algumas das classes que estão incluídas no módulo Data Model, todas elas relacionadas com entidades que suportam auditoria de dados. Sempre que é detectado que uma entidade vai ser inserida ou actualizada (o mecanismo de detecção encontra-se implementado pelo módulo Audit e baseia-se na API de `Interceptors` do Hibernate), são previamente actualizados os atributos de auditoria dessa entidade – data de

inserção/actualização e o identificador do utilizador que realiza a operação, dado que pode ser obtido a partir do serviço `MembershipService`, descrito na secção 3.1.4.

3.1.3. Data UI

O módulo Data UI implementa de forma genérica a View e o Controller do padrão de desenho Model-View-Controller (MVC). Com este módulo, pretende-se reduzir o esforço e tempo necessário para a implementação de formulários de dados, dado que sua estrutura foi pensada e desenhada de forma a integrar-se com o módulo Data. O diagrama de classes da figura 18 mostra, entre outros elementos, a classe abstracta `AbstractDataController`, a partir da qual todos os restantes Controllers devem herdar, dando assim a possibilidade de redefinir a implementação genérica. Esta classe define um construtor que recebe como parâmetro obrigatório um Repositório, que é injectado automaticamente recorrendo às funcionalidades implementadas no módulo Inversion of Control e para o qual são delegadas as acções que requeiram interacção com a BD. Dado que o acesso ao Repositório é feito a partir do Controller, este também é responsável por iniciar e finalizar uma Unidade de Trabalho.

A implementação genérica de uma View do padrão MVC, representada pela classe `DataViewTopComponent`, integra este módulo no sistema de janelas da plataforma NetBeans (dado descender de `TopComponent`, uma classe da plataforma usada especificamente para o efeito). Tem como funcionalidades:

- Permitir obter uma colecção com todos os registos (objectos) de uma entidade;
- "Navegar" na colecção de registos;
- Visualizar os dados de um registo;
- Adicionar e preencher os dados de novos registos;
- Eliminar um registo;
- Actualizar os dados de um registo, de forma bidireccional (da aplicação cliente para a base de dados e vice-versa);

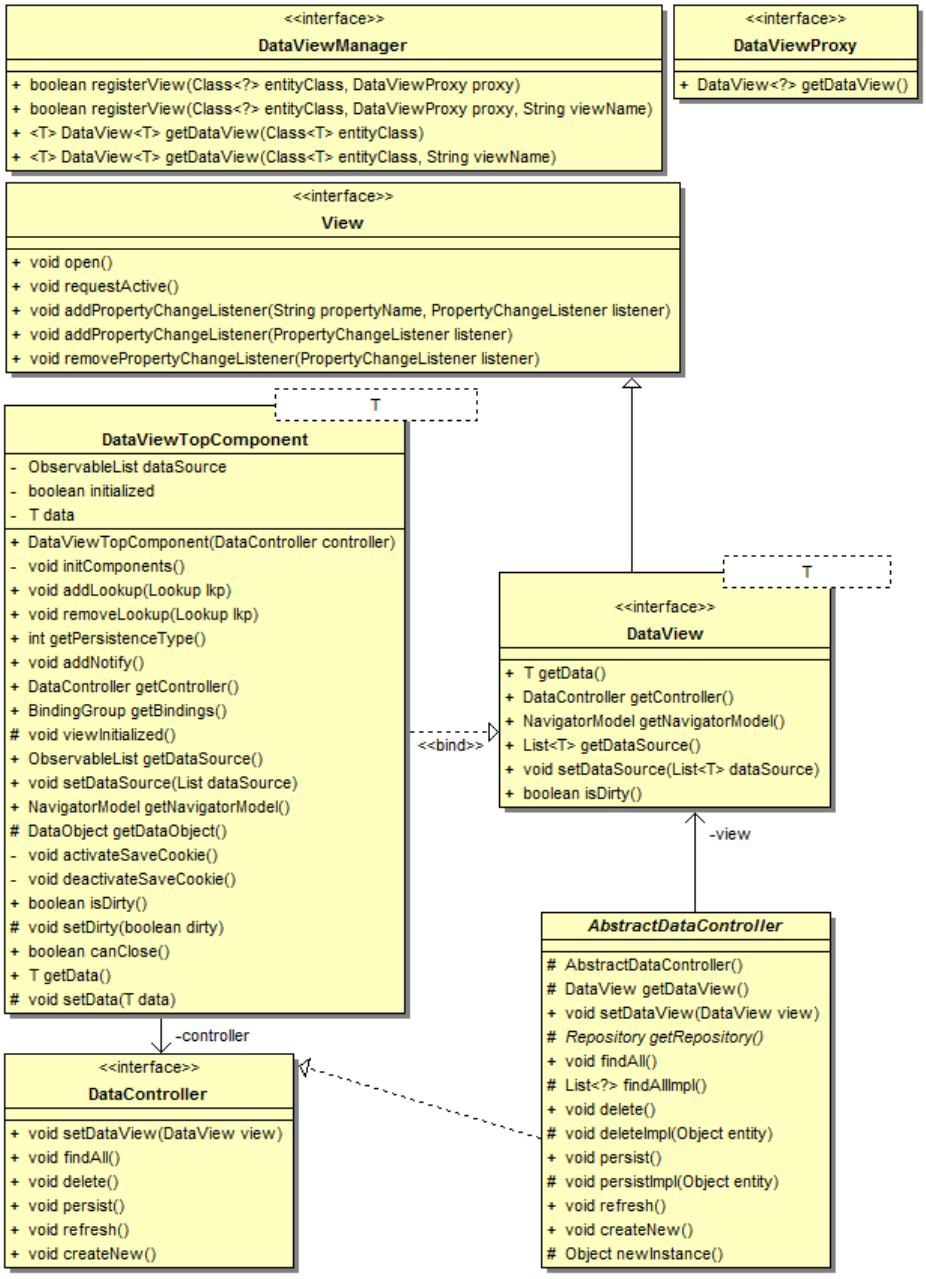

Figura 18 - Diagrama de classes da implementação do padrão de desenho MVC

A figura 19 mostra uma `JToolBar` composta por um conjunto de botões que permitem aceder às funcionalidades disponibilizadas pela classe `AbstractDataController`.

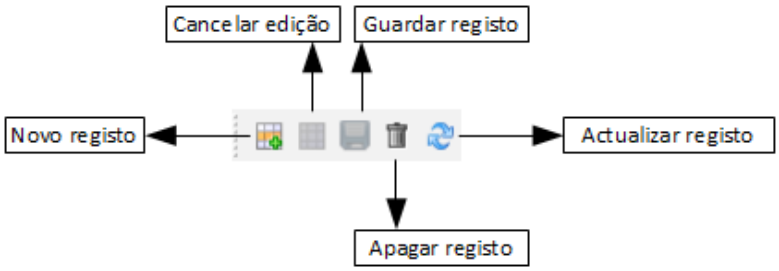

Figura 19 - Descrição dos botões da barra de edição

As funcionalidades relativas à navegação de dados encontram-se descritas no diagrama de classes da figura 20. Este modelo foi implementado seguindo as práticas aplicadas para os restantes componentes existentes no Swing da plataforma JAVA, que também são definidos segundo o padrão de desenho MVC. De um modo geral, os componentes Swing são criados através do seguinte procedimento:

1. Criar uma interface que define o contrato geral do modelo. O nome da interface deve ter como sufixo a palavra "Model";

2. Criar uma classe abstracta com funcionalidades comuns a todas as implementações do modelo. Esta classe deve implementar (parcialmente) a interface definida no ponto anterior. O nome da classe deve ter como prefixo a palavra "Abstract";

3. Criar uma classe que representa uma implementação predefinida do modelo. Esta classe deve herdar da classe abstracta definida no ponto anterior. O nome da classe deve ter como prefixo a palavra "Default";

4. Criar a classe que implementa o componente visual propriamente dito. Neste caso em concreto, a classe herda de `JToolBar` e implementa a interface `ChangeListener`. Os objectos desta classe devem registar-se a uma instância do modelo e ficar à escuta de eventuais alterações do seu estado. Assim que uma alteração é detectada, o componente deverá realizar os passos necessários para representar visualmente essas alterações, mantendo-se assim sincronizado com o estado actual do modelo.

De uma forma geral, o modelo de navegação de dados implementa o conceito de navegação sobre uma lista de objectos, onde é incorporada a noção de elemento seleccionado (ou índice

seleccionado). A partir deste elemento, é possível navegar para o elemento anterior e seguinte, assim como, a qualquer momento, navegar para o primeiro ou último elemento da lista.

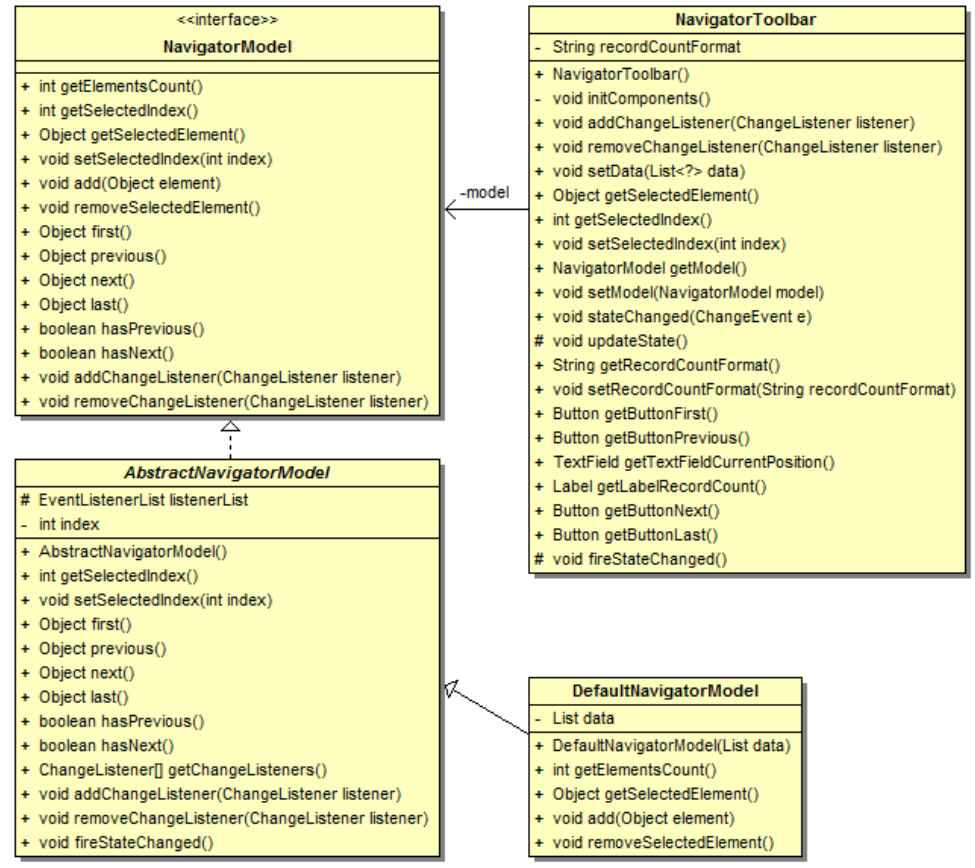

Figura 20 - Diagrama de classes do modelo de navegação

A figura 21 mostra a implementação de uma vista de um modelo de navegação, representada através de uma `JToolBar`.

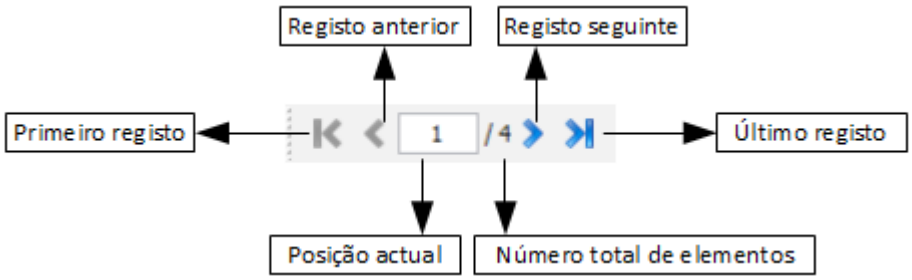

Figura 21 - Descrição dos botões da barra de navegação

3.1.4. Membership

Este módulo define o modelo de gestão de utilizadores e encontra-se descrito no diagrama de classes da figura 22. Actualmente apenas define uma entidade que contém os dados acerca de um utilizador, dos quais se destacam o nome do utilizador e a sua palavra-chave. Estes dados representam as credenciais do utilizador e são necessários para a sua autenticação, processo levado a cabo pelo `MembershipService` e que tem de ser bem sucedido para se poder ter acesso à janela principal da aplicação. Este serviço é registado no contentor de serviços descrito na secção 3.1.1., de forma a ser possível obter os dados sobre o utilizador que actualmente utiliza a aplicação em qualquer ponto do código, através do método `getCurrentUser()`.

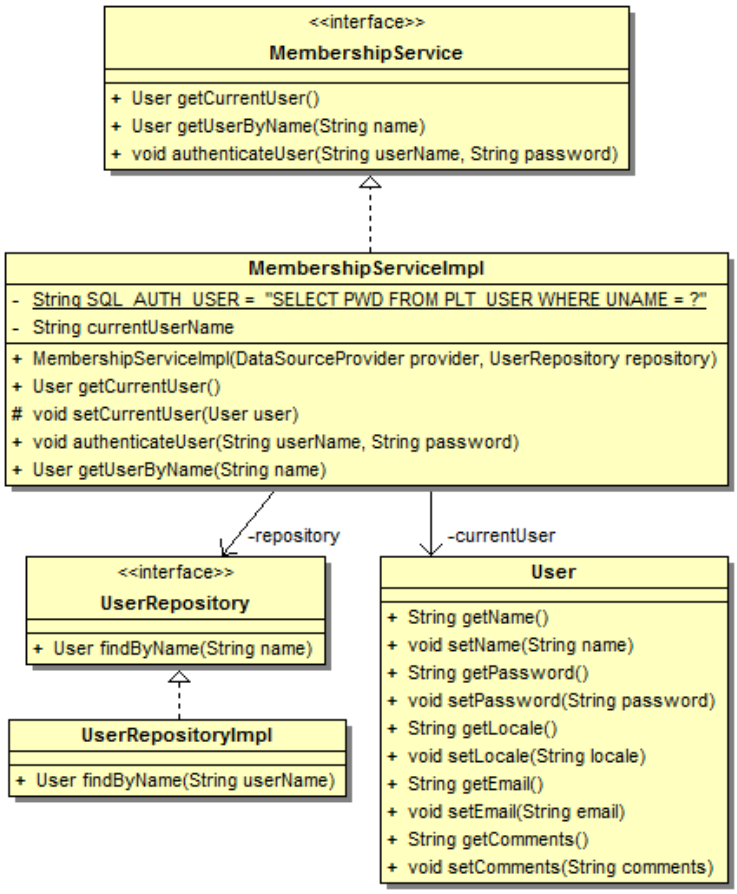

Figura 22 - Diagrama de classes do modelo de gestão de utilizadores

A janela de autenticação, apresentada na figura 24, é mostrada imediatamente antes de ser apresentada a janela principal da plataforma NetBeans e permite ao utilizador indicar as suas

credenciais, de modo a proceder à sua autenticação. Permite também configurar os parâmetros de ligação ao servidor J2EE, cujo papel, nesta situação em particular, é simplesmente fornecer os dados necessários para o estabelecimento da ligação à BD. Estes dados são obtidos através da invocação remota de um Enterprise JavaBean (EJB) que implementa a interface `DataSourceDescriptor`, descrita no diagrama de classes da figura 23.

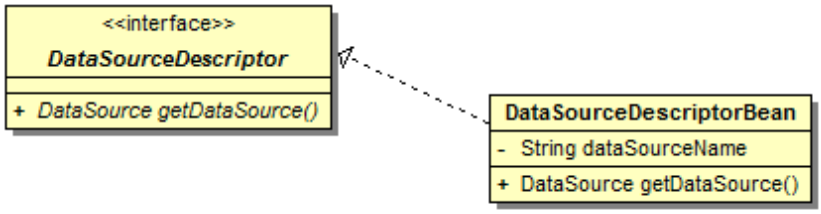

Figura 23 - Definição e implementação da interface DataSourceDescriptor

A utilização de um EJB para acesso aos parâmetros de ligação à BD tem como objectivos:

- **Centralizar a administração da ligação à BD** – O administrador tem a capacidade de alterar os parâmetros de ligação à BD numa única localização (o servidor), sendo que essas alterações serão propagadas para todas as aplicações-cliente da próxima vez que forem executadas;

- **Dificultar o acesso aos dados da ligação à BD** – Dado que as aplicações-cliente não gerem por elas próprias os dados da ligação à BD, torna-se assim mais difícil executar um ataque ao SGBD, o que por sua vez torna o sistema mais seguro.

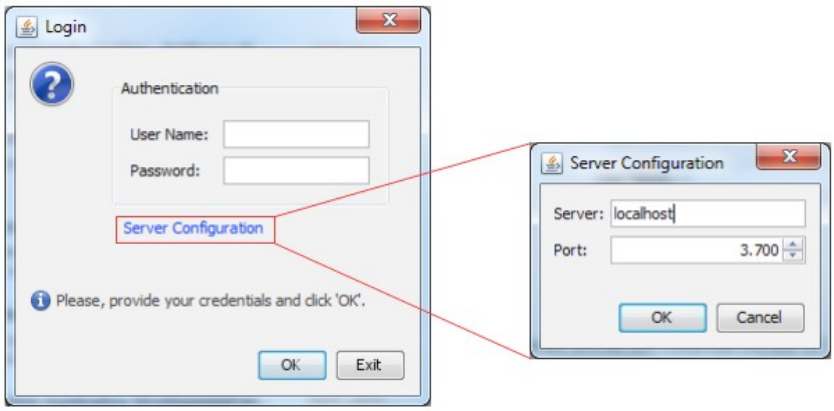

Figura 24 - Janela de autenticação e de configuração da ligação ao servidor

3.1.5. Geometry Model

O modelo de geometrias foi definido com base na informação recolhida acerca do modelo de dados utilizado pelo OpenStreetMap e tem, como propósito, possibilitar a georreferenciação das estruturas marítimas, incluindo os seus troços. Pretende-se, contudo, que este modelo seja compatível com o Hibernate, visto ser esta a biblioteca escolhida para a implementação da camada de Acesso a Dados. Por isso, utilizou-se a extensão HibernateSpatial para mapear tipos de dados geográficos específicos a cada SGBD para uma `Geometry` da biblioteca JTS. A classe `GeometryEntity`, apresentada no diagrama de classes da figura 25, representa um objecto geométrico que se liga à classe `GeometryTag`. Esta corresponde a uma Etiqueta, conceito definido na secção 2.3.2.1., referente ao modelo de dados do OpenStreetMap.

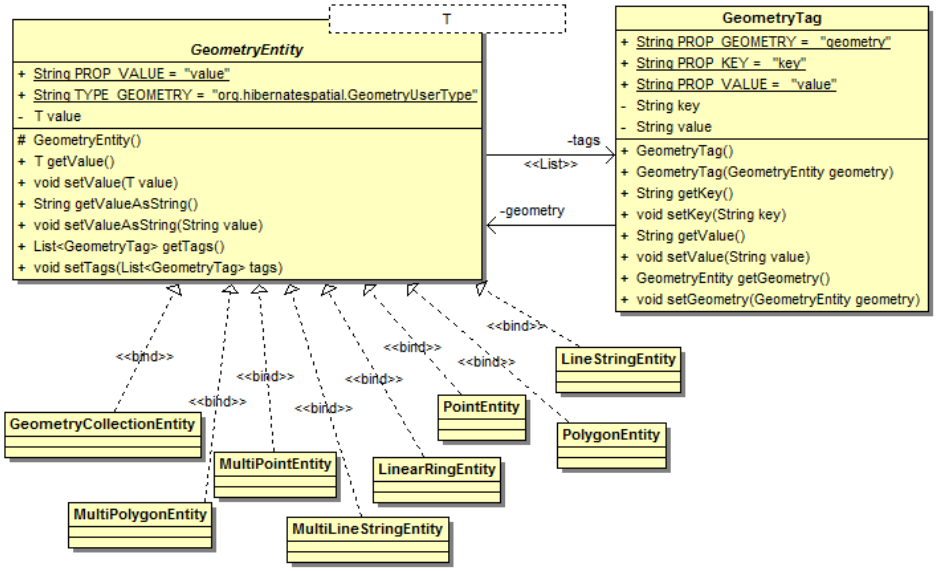

Figura 25 - Diagrama de classes do modelo de geometrias

3.1.6. Inspections Model

O modelo de inspecções pode ser dividido em dois sub-modelos interligados:

- **Modelo da estrutura de uma ficha de inspecção** – é definida por um questionário (`Questionnaire`) composto por um conjunto de perguntas (`Question`) que podem ser organizadas em grupos e sub-grupos (`QuestionGroup`) e, para cada uma, existe um conjunto de respostas possíveis (`AnswerValue`). Uma resposta é definida pelo seu tipo de dados associado (`AnswerType`), nomeadamente, booleano,

número, texto, data ou enumeração e pelo valor da sua avaliação (`AnswerEvaluation`), como por exemplo, resposta certa ou errada, nível de danos fraco, médio ou elevado, etc. A uma inspecção está sempre associado outro objecto susceptível de ser inspeccionado. Dado que, conceptualmente, qualquer objecto pode ser submetido a uma inspecção, a noção de objecto "inspeccionável" foi definida através de uma interface denominada `Inspectable`. A figura 26 apresenta o diagrama de classes que representa este modelo;

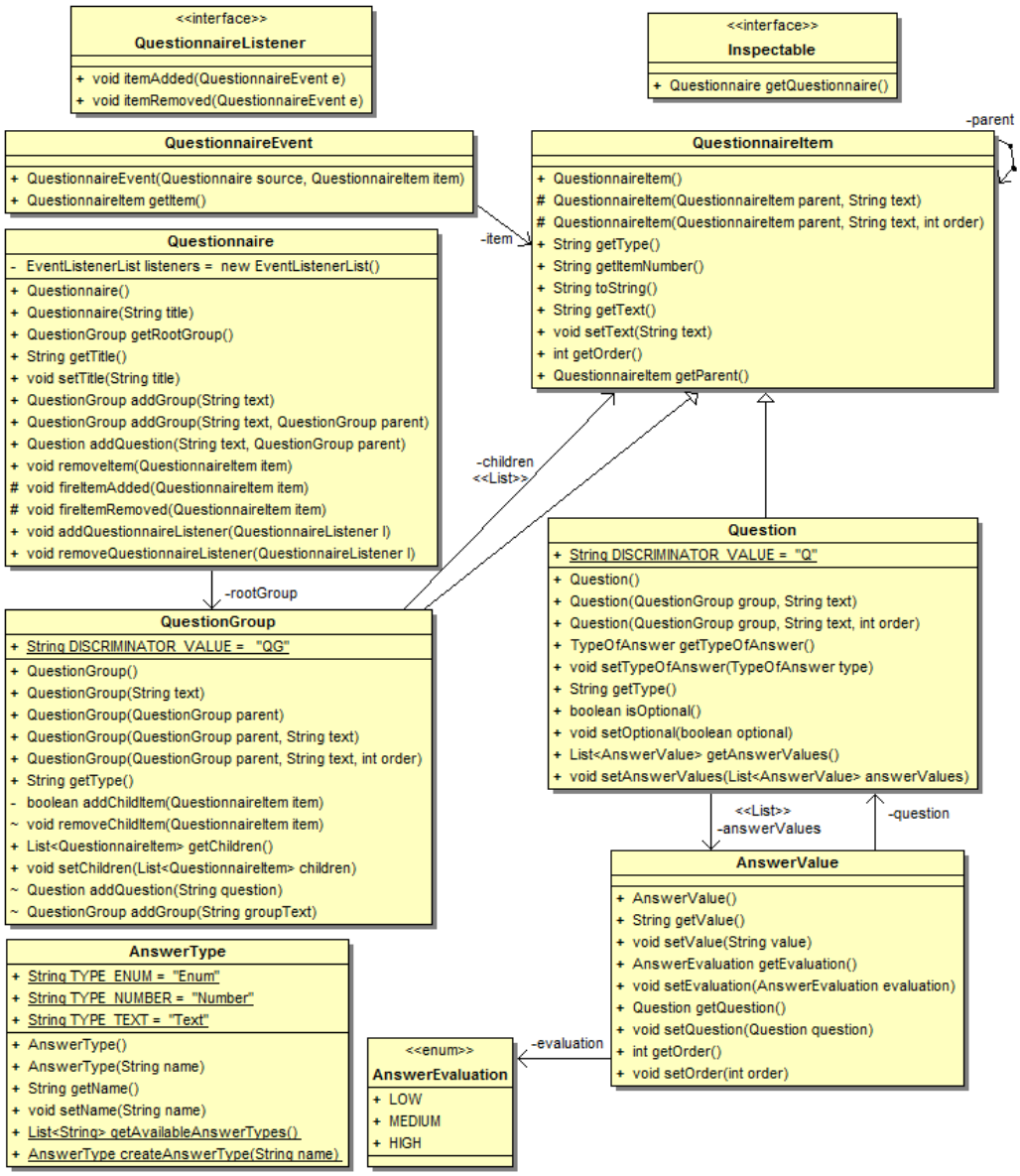

Figura 26 - Diagrama de classes do modelo de fichas de inspecção

- **Modelo de fichas de inspecção preenchidas** – pode considerar-se como sendo uma instância de uma ficha de inspecção (`AnsweredQuestionnaire`), realizada por um utilizador numa determinada data e local, e que contém as respostas (`Answer`) dadas a cada pergunta. Deve ser possível avaliar uma ficha de inspecção preenchida, analisando o valor de avaliação de cada resposta dada. O diagrama de classes da figura 27 apresenta as classes que constituem este modelo.

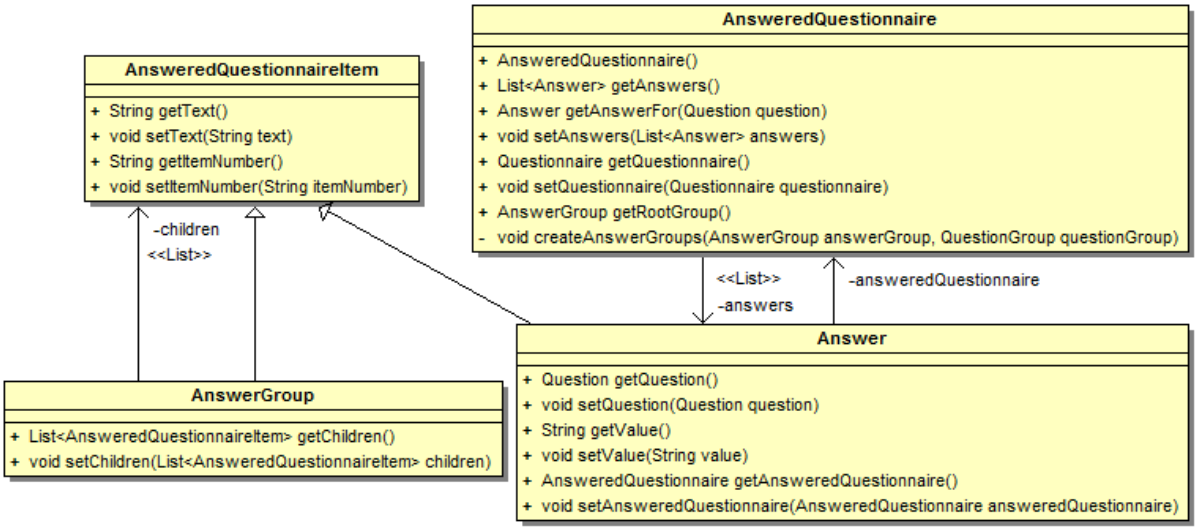

Figura 27 - Diagrama de classes do modelo de fichas de inspecção preenchidas

Dado que o utilizador poderá ter de definir uma estrutura hierárquica de grupos e sub-grupos para criar uma ficha de inspecção, foi decidido utilizar um componente em árvore, de modo a tornar a representação gráfica consistente com a natureza do modelo. O componente utilizado é um objecto da classe `OutlineView`, definida na Explorer & Property Sheet API da plataforma NetBeans, e combina as funcionalidades de uma `JTree` com uma `JTable` (formando um componente comummente conhecido por TreeListView). A figura 28 apresenta o formulário para criação de uma ficha de inspecção, onde se pode constatar o aspecto de um componente deste tipo. É possível também observar os botões à esquerda da imagem, que representam as acções de edição de uma ficha de inspecção, designadamente, adicionar um novo grupo, adicionar uma nova questão e remover um item (ou um nó) do questionário. O estado activo de cada botão é determinado pelo item da árvore que se encontra seleccionado.

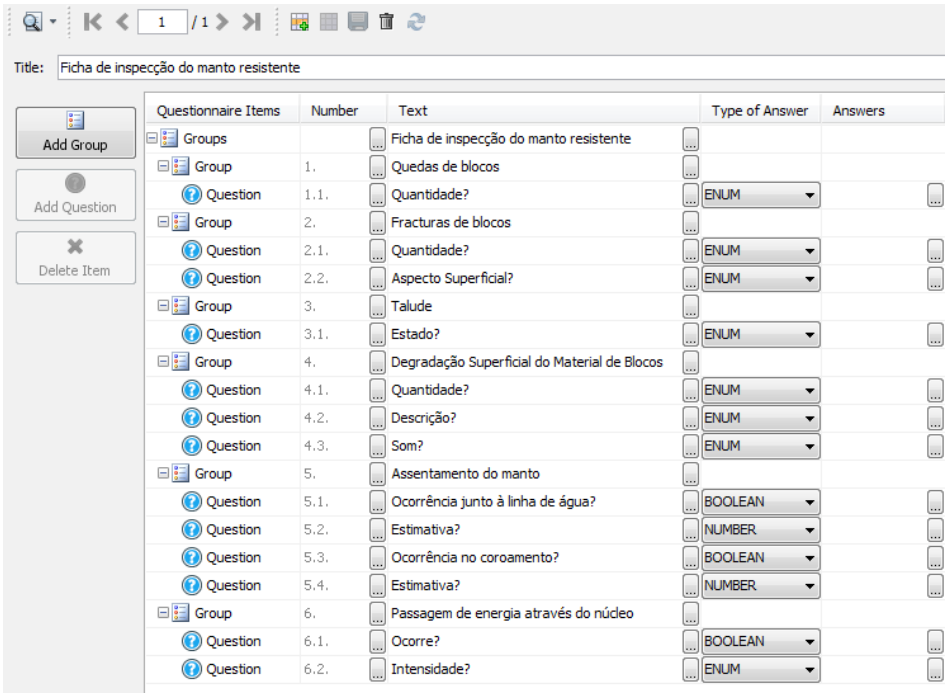

Figura 28 - Janela de gestão de fichas de inspecção

O componente `OutlineView` tem a capacidade de utilizar objectos do tipo `PropertyEditor`, cuja classe se encontra definida na JavaBeans API da platforma JAVA, para edição de dados. Estes objectos permitem editar de forma mais adequada o tipo de dados das células de uma coluna, utilizando para isso um componente apropriado para o efeito. Por exemplo, na figura 28 é possível observar que a coluna com a designação "Type of Answer" utiliza uma `JComboBox` por cada questão apresentada. Isto deve-se ao facto dessa propriedade ser uma enumeração e, por omissão, o `PropertyEditor` que representa visualmente este tipo de dados utiliza `JComboBox`es preenchidas com as designações das enumerações. É, no entanto, possível (re)definir a associação entre a implementação de um `PropertyEditor` e um tipo de dados, efectuando o seu registo a partir da classe `PropertyEditorManager`, indicando para isso a classe que representa o `PropertyEditor` e a classe que representa o tipo de dados que se pretende gerir. Este registo deve ser realizado numa fase inicial da execução da aplicação.

Para se definir as respostas possíveis a uma pergunta, utilizando as células da coluna "Answers" para o efeito, decidiu-se implementar um `PropertyEditor` que permita realizar, de forma genérica, a edição de propriedades de objectos que representam colecções.

A figura 29 apresenta o diagrama de classes do modelo, de onde se destacam as classes `AbstractCollectionPropertyEditor` e `AnswerValuesEditor`. A primeira é a implementação base para qualquer `PropertyEditor` de colecções. A segunda é a implementação específica de um editor de colecções de objectos do tipo `AnswerValue`, a classe que representa uma resposta possível. Esta implementação é necessária de modo a especificar qual a classe que define a colecção (por exemplo, `ArrayList`, `Vector`, etc) e qual a classe que define cada um dos elementos da colecção (neste caso tratam-se de objectos da classe `AnswerValue`).

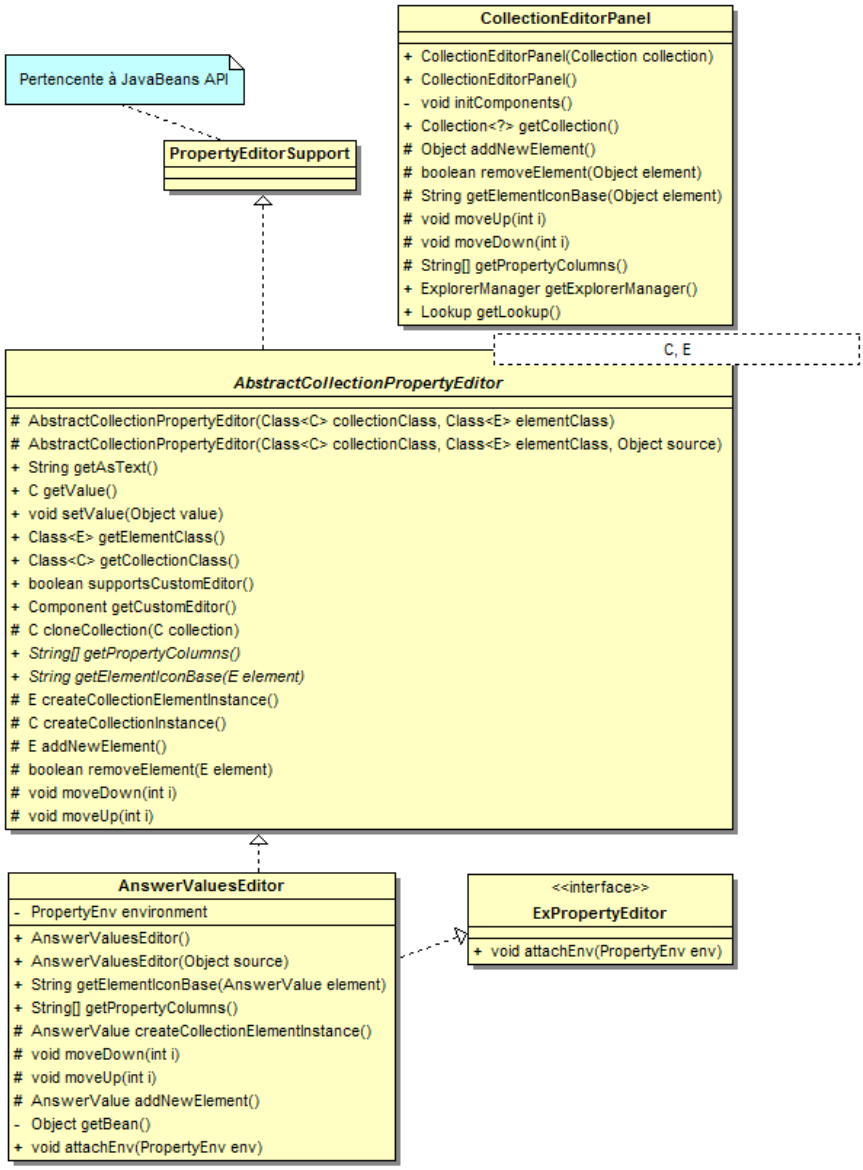

Figura 29 - Modelo de PropertyEditors para colecções de objectos

A figura 30 mostra a janela para edição de colecções, implementada recorrendo ao modelo descrito anteriormente. Esta janela é apresentada sempre que o utilizador carrega no botão disponibilizado em cada célula da coluna "Answers" da figura 28. A sua estrutura é sempre igual para qualquer colecção de objectos, sendo que a parte variável é aquela que é apresentada ao lado direito da janela e que diz respeito às propriedades do objecto seleccionado à esquerda.

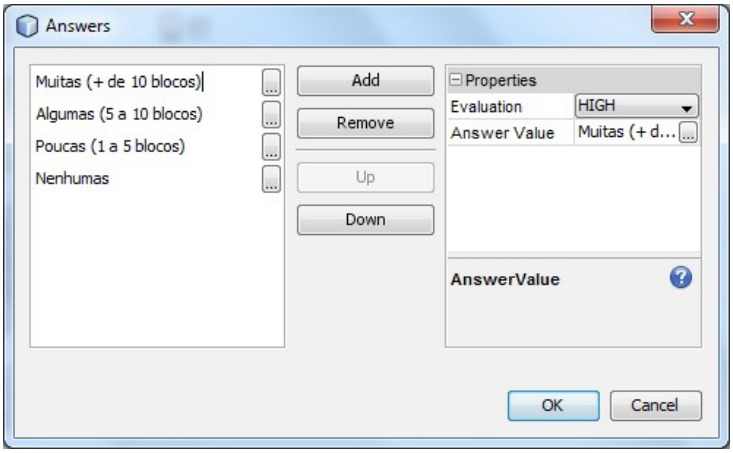

Figura 30 - Janela de gestão de respostas a uma pergunta

3.1.7. Structures Model

Tal como foi referido na secção anterior, uma inspecção tem sempre associada um objecto susceptível de ser inspeccionado – um objecto "inspeccionável" – que se define através da interface `Inspectable`. No caso da monitorização de obras marítimas, os objectos que são submetidos a inspecções são os elementos estruturais, que estão representados pela classe `StructureElement` no diagrama de classes da figura 31. Este modelo servirá para caracterizar as obras existentes num porto (`Port`), mas actualmente, apenas define a estrutura necessária para a implementação do modelo de fichas de inspecção. Assim, considera-se que uma estrutura marítima (`Structure`) faz parte de um porto e pode ser dividida em troços ou partes (`StructurePart`), sendo que cada uma destas partes tem associada os elementos estruturais que a constituem.

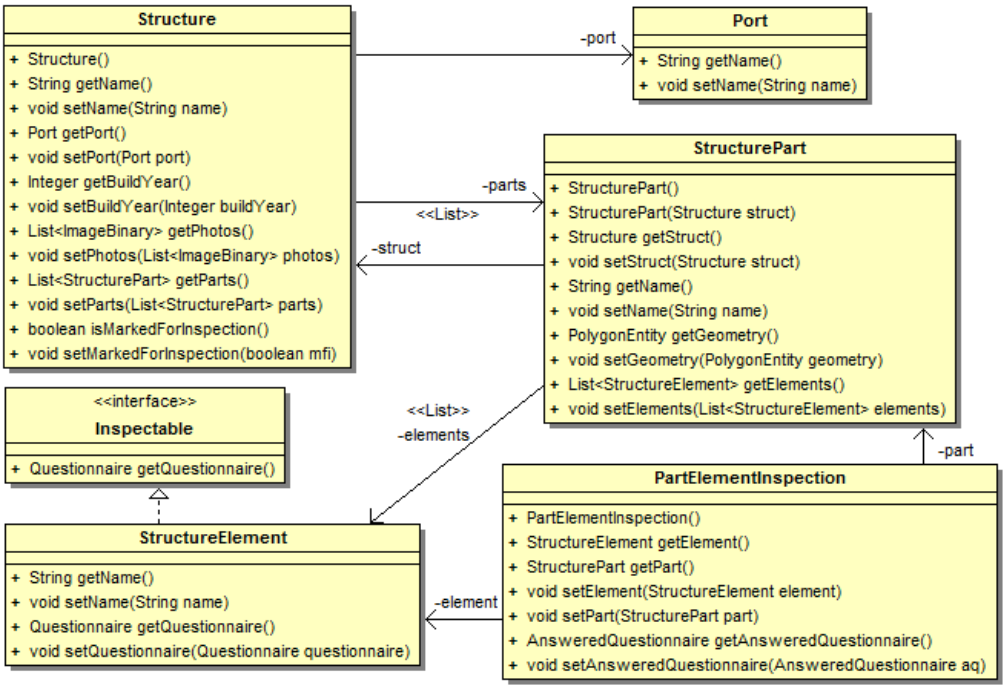

Figura 31 - Diagrama de classes do modelo de obras marítimas

O utilizador pode marcar ou assinalar uma ou mais estruturas para inspecção, de modo a definir que dados de estruturas estarão disponíveis para as aplicações móveis, a fim de se proceder à realização das respectivas inspecções. A figura 32 apresenta a janela que permite caracterizar uma obra marítima, onde também é possível marcar ou desmarcar a obra através da `JCheckBox` designada por "Marked for inspection".

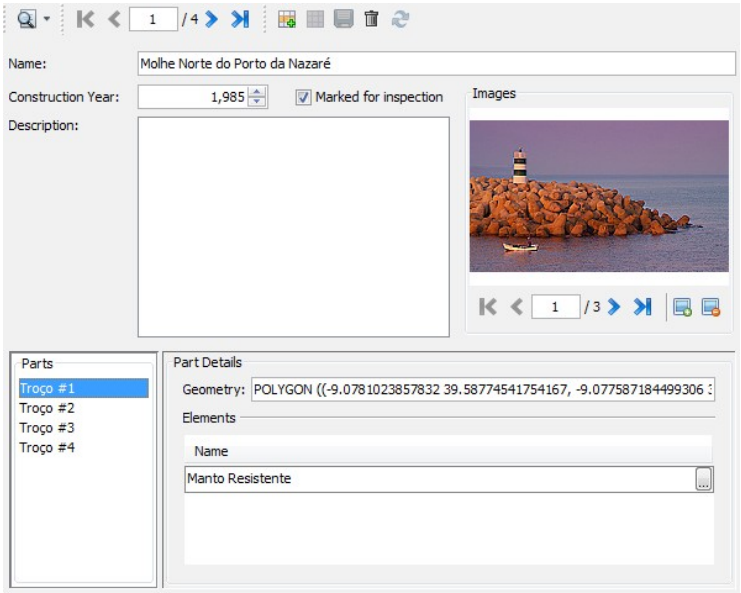

Figura 32 - Janela de gestão de estruturas de um porto

A classe `PartElementInspection` permite registar cada uma dessas inspecções, associando um par elemento estrutural/troço a uma ficha de inspecção preenchida. A informação que consta nas fichas de inspecção preenchidas podem ser consultada através da janela apresentada na figura 33.

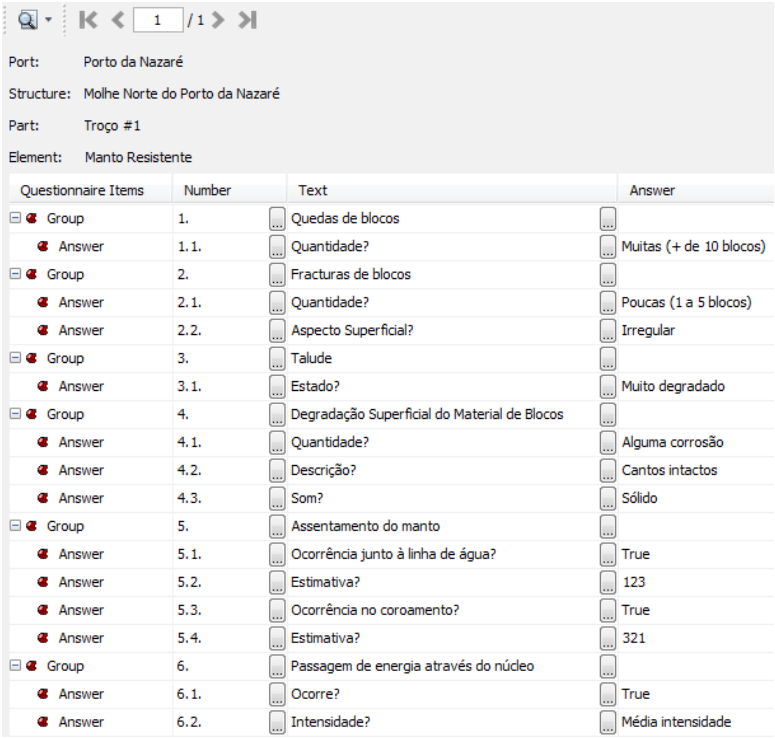

Figura 33 - Janela de visualização de fichas de inspecção preenchidas

3.2. Arquitectura da aplicação servidor

A aplicação servidor é baseada nas tecnologias J2EE e SOAP Web Services. Como servidor aplicacional foi utilizado o Glassfish, por ser gratuito e por ser possível desenvolver, instalar, depurar e testar o código da aplicação directamente a partir do NetBeans IDE. A figura 34 apresenta a visão geral dos módulos utilizados ou implementados para o desenvolvimento desta aplicação. Note-se que alguns dos módulos indicados na figura são os mesmos que foram descritos na secção relativa à arquitectura de alto nível da aplicação *desktop*, como é o caso do módulo Data. É preciso ter em consideração que algumas das funcionalidades implementadas neste módulo não serão utilizadas, nomeadamente a gestão de transacções (implementada através do padrão de desenho Unit of Work, tal como tinha sido referido anteriormente), dado que o próprio Glassfish já as disponibiliza. Daí a importância do módulo

Data separar esta gestão de transacções do acesso aos dados propriamente dito. Deste modo, torna-se possível retirar por completo as funcionalidades associadas a essa gestão, reaproveitando o código relativo à implementação dos Repositórios.

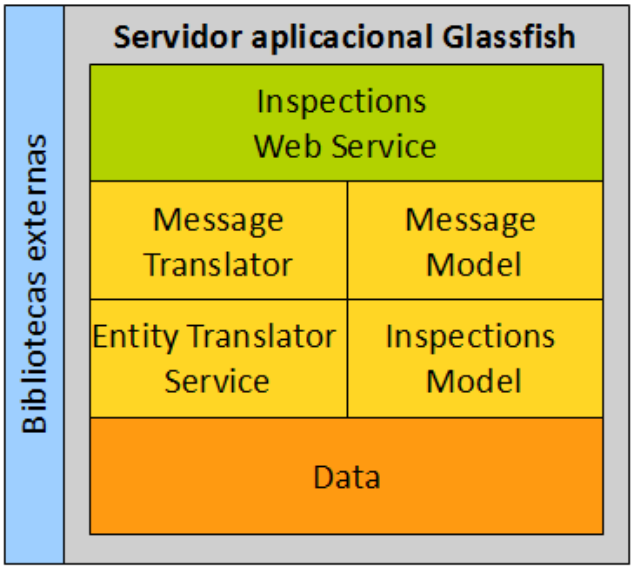

Figura 34 - Visão geral dos módulos da aplicação servidor

De seguida são descritos em detalhe as partes constituintes da aplicação servidor, composta pelos seguintes módulos:

- **Data** – Este módulo é o mesmo que foi utilizado para a aplicação *desktop* e encontra-se descrito na secção 3.1.2.;
- **Inspections Model** – Outro módulo que foi implementado para a aplicação *desktop*, cuja descrição se encontra na secção 3.1.6.;
- **Entity Translator** e **Message Translator**– Implementação do padrão de desenho Entity Translator;
- **Message Model** – Modelo que define a estrutura da mensagem que é trocada entre a aplicação servidor e as aplicações móveis;
- **Inspections Web Service** – Um serviço que permite o acesso remoto aos dados das estruturas marítimas e respectivas fichas de inspecção. Permite também o envio dos dados relativos às fichas de inspecção preenchidas;

3.2.1. Entity Translator (Tradutor de Entidades)

O Tradutor de Entidades é um padrão de desenho que tem como principal objectivo transformar um modelo de objectos noutro [17]. A utilização de um padrão desta natureza é ditada pela necessidade de estabelecer um canal de comunicação entre a aplicação *desktop* e a aplicação móvel. Ambas têm modelos de objectos que apresentam algumas diferenças, quer ao nível dos relacionamentos, quer ao nível dos atributos dos objectos que compõem esses modelos. Acresce também o facto de que a mensagem transmitida entre as aplicações é definida através de outro modelo de objectos distinto, pensado e desenhado de forma a optimizar a quantidade de dados a enviar pela rede. A aplicação deste padrão de desenho permite atingir os seguintes objectivos:

- Isolar os modelos de objectos, permitindo que sejam efectuadas alterações nos mesmos sem que entrem em conflito uns com os outros. Esta separação impede a propagação de erros entre as aplicações;
- Evita a utilização de regras de serialização em modelos que não foram pensados para serem utilizados em Web Services;
- Permite realizar de forma independente o desenvolvimento da aplicação *desktop* e da aplicação móvel.

A principal desvantagem deste esquema prende-se com o facto de se passar a ter mais uma base de código para gerir – aquela que diz respeito à camada que efectua a tradução propriamente dita. Contudo, neste caso em concreto os benefícios são maiores que os custos, dado tratar-se de uma camada fina, contendo uma quantidade reduzida de lógica (em comparação com os restantes modelos), sendo que todo o código se concentra numa só classe – a classe responsável por traduzir de um modelo de objectos para outro.

A implementação deste padrão de desenho foi retirada e adaptada para a linguagem JAVA a partir da plataforma Smart Client Software Factory, desenvolvida em C# e VB.NET pela Microsoft, e o seu modelo descreve-se segundo o diagrama de classes da figura 35.

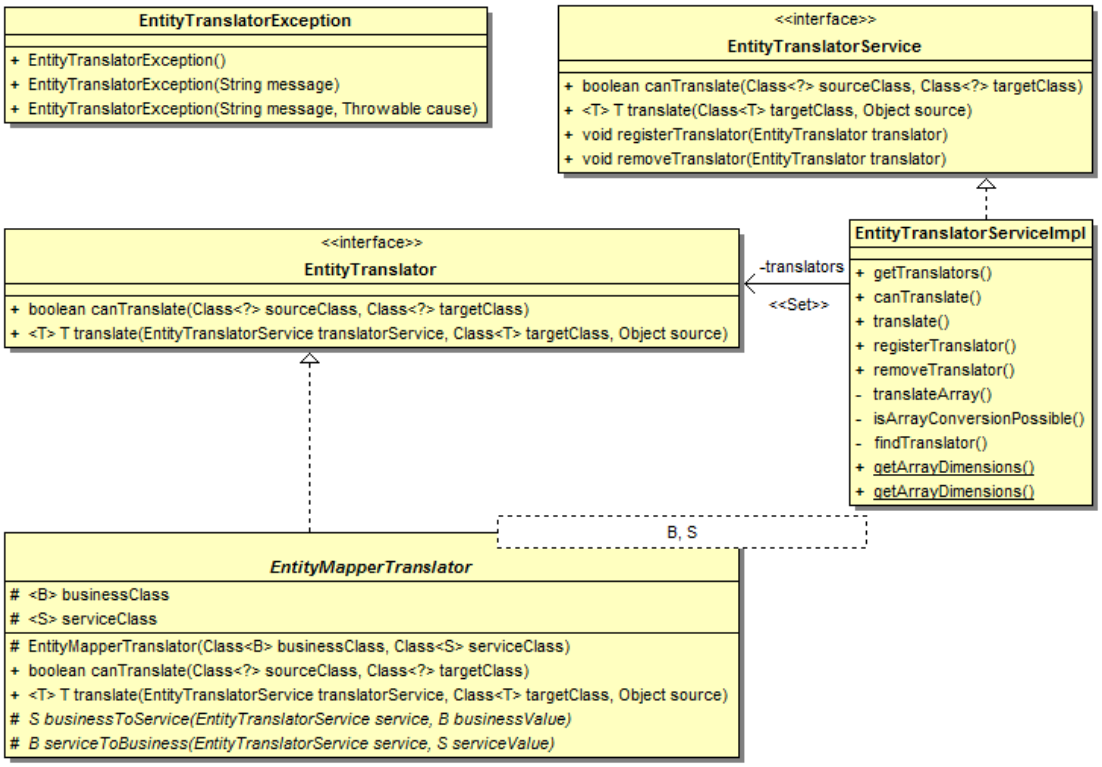

Figura 35 - Diagrama de classes do padrão de desenho Entity Translator

Do diagrama da figura 35 destacam-se três entidades:

- **EntityTranslator (Tradutor)** – Interface que define o contrato a implementar respeitante à tradução de uma instância de uma classe para outra de uma classe diferente;

- **EntityMapperTranslator (Tradutor Especializado)** – Classe abstracta a partir da qual os restantes Tradutores devem herdar. Divide o processo de tradução em duas tarefas, de acordo com a classe origem (a classe a traduzir) e a classe destino (a tradução), através da implementação dos métodos `businessToService()` e `serviceToBusiness()`;

- **EntityTranslatorService (Serviço de Tradução)** – Interface que define um ponto único para a utilização do padrão. Tem dupla funcionalidade: permite registar um ou mais Tradutores e traduz um objecto de uma classe para um objecto de outra classe. O processo de tradução é delegado para um Tradutor que seja capaz de lidar com esse tipo de objectos.

Para utilizar este padrão é necessário realizar três passos:

1. Implementar um Tradutor, que normalmente herda de um Tradutor Especializado;

2. Instanciar um Serviço de Tradução e registar uma nova instância do Tradutor implementado no ponto anterior. Este passo é normalmente realizado na parte de código relativo à inicialização de uma aplicação. Deve realçar-se o facto que normalmente há apenas uma instância do Serviço de Tradução por aplicação. Poderá ser implementado aplicando o padrão de desenho Singleton (não recomendado) ou recorrendo a um contentor de serviços, tal como aquele que foi implementado para o módulo Inversion of Control;

3. Realizar a tradução propriamente dita, na parte do código onde esta é necessária. Para tal, basta obter a instância do Serviço de Tradução e invocar o método `translate(Class, Object)`. Em caso de sucesso, o valor de retorno deste método será uma nova instância de outra classe resultante do processo de tradução.

Para o projecto de monitorização de obras marítimas foram implementadas classes que representam o Tradutor e as Mensagens que são necessárias trocar entre as várias aplicações, descritas no diagrama de classes da figura 36. O Tradutor implementado tem a capacidade de traduzir de um objecto do tipo `ModelMessage` para um objecto do tipo `InspectionsMessage` e vice-versa. Uma `ModelMessage` conhece os objectos do modelo de inspecções e de estruturas e é constituída por:

- Um conjunto de estruturas que foram assinaladas para inspecção (caso se trate de um pedido de dados de uma aplicação-cliente);

- Um conjunto de fichas de inspecção preenchidas (caso se trate de um envio de dados de uma aplicação-cliente);

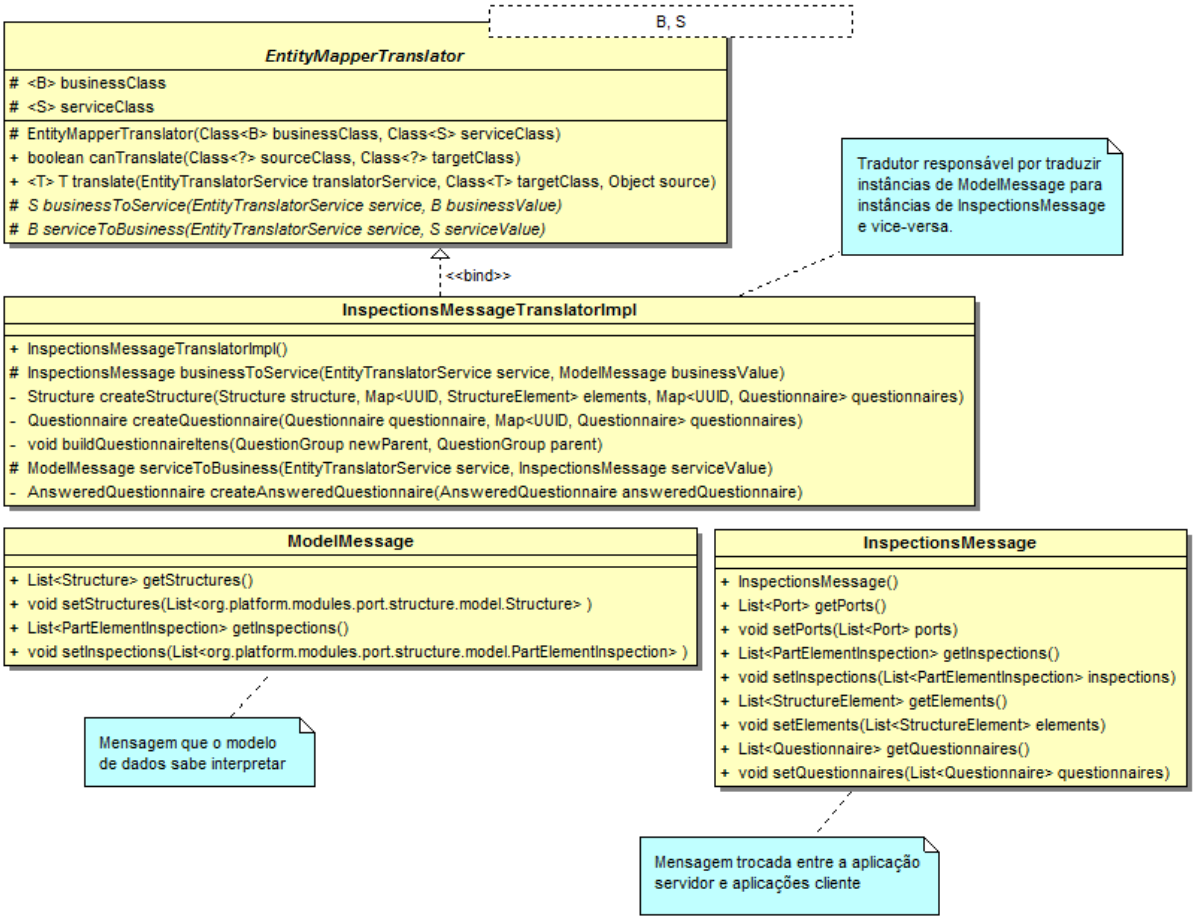

Figura 36 - Implementação do padrão de desenho Entity Translator

Os objectos do tipo `InspectionsMessage` ligam-se a um modelo implementado especificamente para a transmissão de dados entre as aplicações-cliente e o servidor e descreve-se segundo o diagrama de classes da figura 37. Este modelo foi pensado de forma a:

- Eliminar relacionamentos inválidos (relacionamentos recursivos) que impedem a serialização dos objectos para XML;
- Optimizar a quantidade de dados transmitidos, evitando redundâncias inerentes à própria estrutura do modelo. Por exemplo, os troços ou partes de uma estrutura podem ser compostas pelos mesmos elementos estruturais que, por sua vez, estão associados às mesmas fichas de inspecção. Se a estrutura original se mantivesse inalterada, a comunicação entre aplicações iria transmitir dados repetidos, encarecendo, por isso, os custos deste processo.

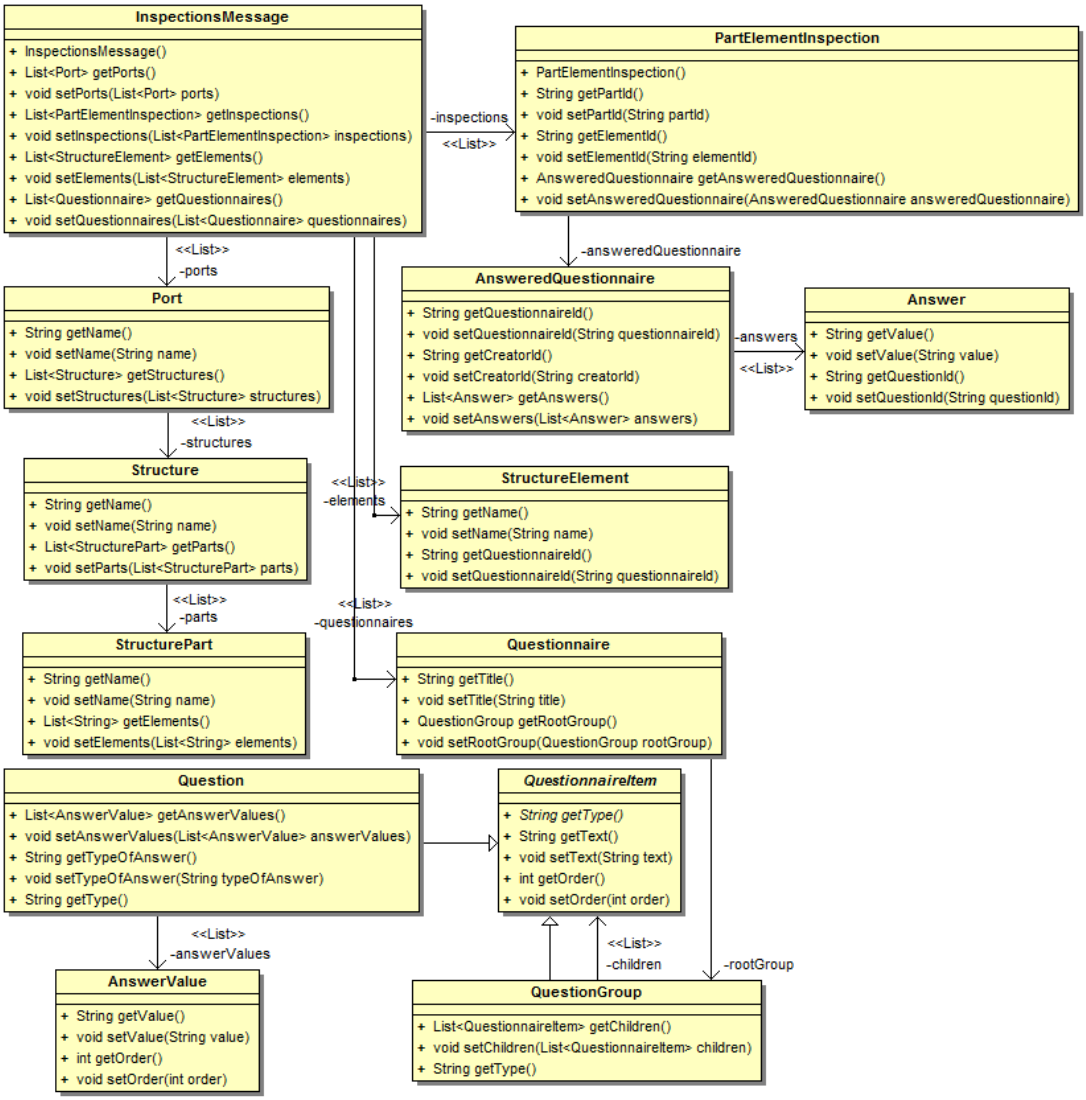

Figura 37 - Diagrama de classes do modelo de mensagens

A figura 38 mostra, de uma forma geral, como se efectua o processo de comunicação entre uma aplicação cliente e a aplicação servidor. O cliente envia o pedido, que poderá incluir uma mensagem, uma instância de `InspectionsMessage` previamente traduzida de uma instância de `ModelMessage`, que, ao ser recebida pelo servidor, é traduzida para uma instância de `ModelMessage` e passada para o modelo de objectos de negócio, onde será decidido o que fazer com os dados nela contidos. Caso seja necessário enviar uma resposta ao cliente, então o processo a realizar é precisamente o inverso. É feita a tradução de uma instância de `ModelMessage` para uma instância de `InspectionsMessage` e é executada a transmissão dessa mensagem.

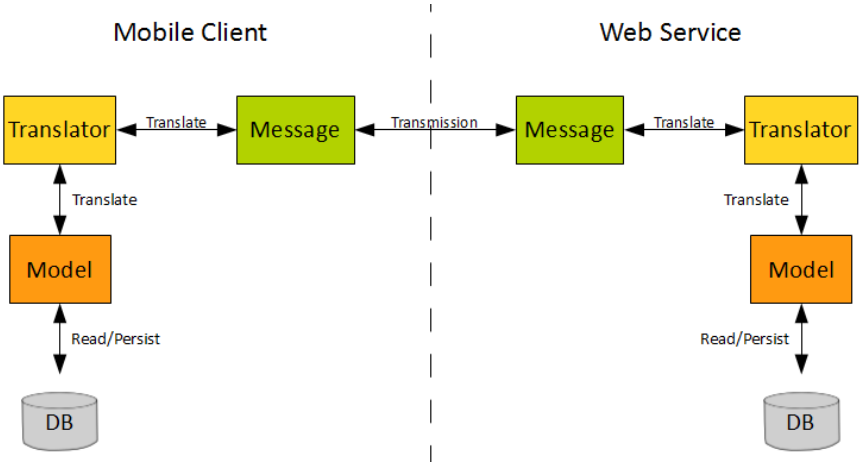

Figura 38 - Transmissão de mensagens entre cliente móvel e Web Service

3.2.2. Descrição detalhada da comunicação entre aplicações

De forma a obter os dados relativos das obras marítimas a inspeccionar, as aplicações cliente devem invocar o método `getInspectionData()` do Web Service. Esses dados incluem informação sobre os troços e seus elementos estruturais e as respectivas fichas de inspecção. O próximo diagrama de sequência mostra os passos necessários para obter a mensagem do serviço:

1. Obtém a lista de obras marítimas marcadas para inspecção a partir do seu repositório (`StructureRepository`);

2. Cria uma `ModelMessage` e atribui-lhe a lista de obras obtida no ponto anterior;

3. Invoca o serviço de tradução de entidades para traduzir de uma `ModelMessage` para uma `InspectionMessage`, o que requer que tenha sido registado um tradutor adequado a essas classes no serviço de tradução. Uma mensagem de inspecções é uma mensagem cuja estrutura foi pensada de modo a optimizar o número de bytes que são transmitidos pela rede. A optimização foi conseguida, definindo um modelo de classes que representa um sub-domínio do modelo original e contém regras de serialização que permitem ajustar a forma como estas classes são transformadas em XML;

4. Envia a `InspectionMessage` produzida no ponto anterior;

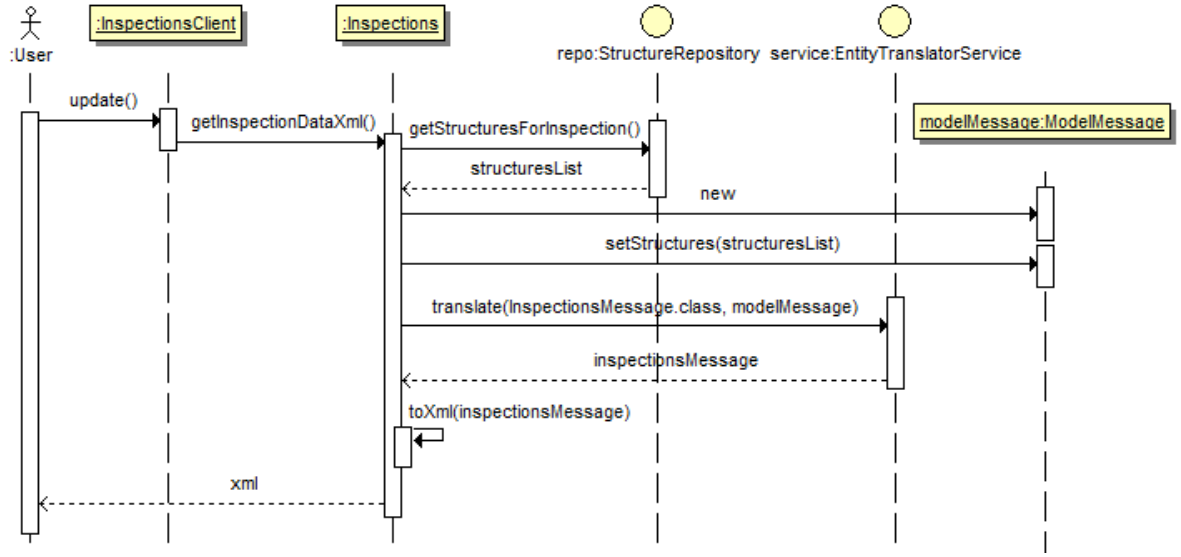

Figura 39 - Diagrama de sequência do processamento de um pedido de dados

Para guardar as fichas de inspecção preenchidas que são enviadas pelas aplicações-cliente, o processo a tomar é o seguinte:

1. Traduz a mensagem de inspecção enviada pela aplicação cliente numa mensagem do modelo;

2. Obtém as fichas de inspecção preenchidas a partir da mensagem do modelo obtida no ponto anterior;

3. Invoca o repositório de dados das fichas preenchidas para persistir os dados destas na base de dados. Deve ser tido em conta que uma ficha de inspecção já com o identificador atribuído poderá ainda não ter sido inserida na base de dados do sistema central;

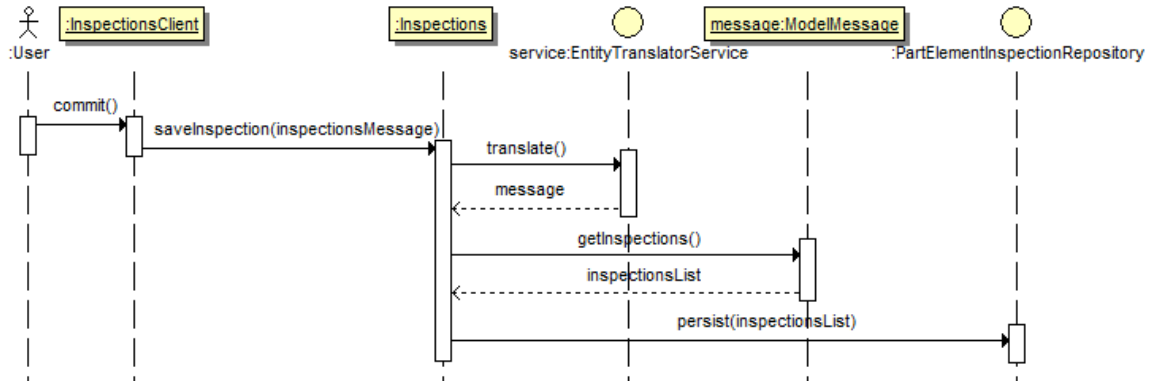

Figura 40 - Diagrama de sequência da persistência de dados na aplicação servidor

3.3. Arquitectura da aplicação móvel

A arquitectura da aplicação móvel foi desenvolvida com base na .NET Compact Framework e tem uma estrutura que apresenta algumas semelhanças com a arquitectura da aplicação *desktop*, que, tal como esta, também se organiza por camadas. Utiliza, inclusivamente, uma implementação do padrão de desenho Inversion of Control, adaptado da biblioteca Compact Container, tal como se indica na figura 41. De realçar também que o módulo Common, contém, entre outras, uma implementação do padrão de desenho Entity Translator, descrito no capítulo 3.2.1.

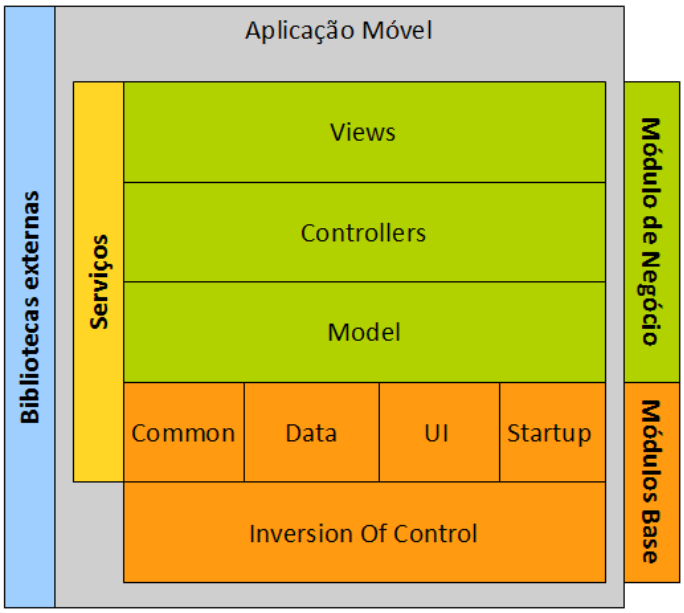

Figura 41 - Visão geral dos módulos da aplicação móvel

As próximas secções descrevem mais aprofundadamente os principais módulos implementados. À semelhança da aplicação *desktop*, os módulos implementados podem ser divididos em duas categorias – os módulos base e os módulos de negócio.

3.3.1. Data

O módulo Data fornece, tal como o seu homólogo da aplicação *desktop*, uma camada lógica para acesso a dados de uma BD recorrendo à utilização de técnicas de O/RM. É uma adaptação de um projecto gratuito e *open-source* denominado sqlite-net [18], que permite:

- Mapear um modelo de objectos a um conjunto de tabelas de uma BD SQLite através de *custom attributes* aplicados às propriedades de cada classe;
- Definir consultas "tipadas" através da utilização de Language-Integrated Query (LINQ).

A adaptação introduzida nesta biblioteca diz respeito a funcionalidades associadas as *triggers* (neste caso, *triggers* aplicacionais) – a cada classe é possível adicionar métodos, com nomes e parâmetros estabelecidos *a priori*, que serão automaticamente invocados em determinados momentos de um acesso a uma BD: depois de realizar uma consulta a um registo ou antes e depois de apagar, inserir ou actualizar um registo. Esta capacidade foi introduzida de modo a facilitar a gestão de relacionamentos entre classes, funcionalidade omissa no código original da biblioteca. Tem, no entanto, a desvantagem de se incluir código de gestão de base de dados nas classes afectas ao modelo de domínio, cujo impacto foi minorado devido à utilização de interfaces, designadamente, a interface `IDatabase`.

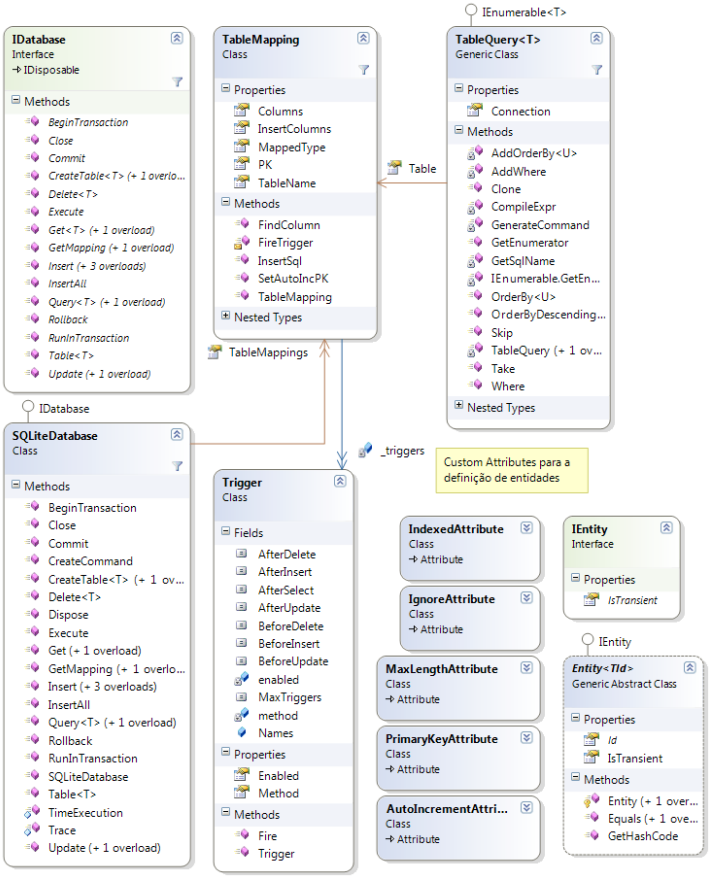

Figura 42 - Diagrama de classes do módulo Data

3.3.2. Model

A grande maioria das classes do modelo de domínio implementado para a aplicação móvel derivam da classe `Entity`, apresentada no diagrama de classes da figura 42, de modo a facilitar a integração do modelo com o módulo de acesso a dados. Este modelo é bastante semelhante ao modelo utilizado na aplicação *desktop* e que foi apresentado nos capítulos 3.1.6. e 3.1.7., relativos ao modelo de inspecções e de estruturas, respectivamente. As diferenças mais relevantes a salientar dizem respeito à inclusão de métodos que representam os *triggers* referidos na secção anterior e também ao número reduzido de propriedades das classes, em comparação com o modelo existente na aplicação *desktop*. Por exemplo, este modelo não contém quaisquer propriedades referentes a dados de auditoria, mencionados na secção 3.1.2.

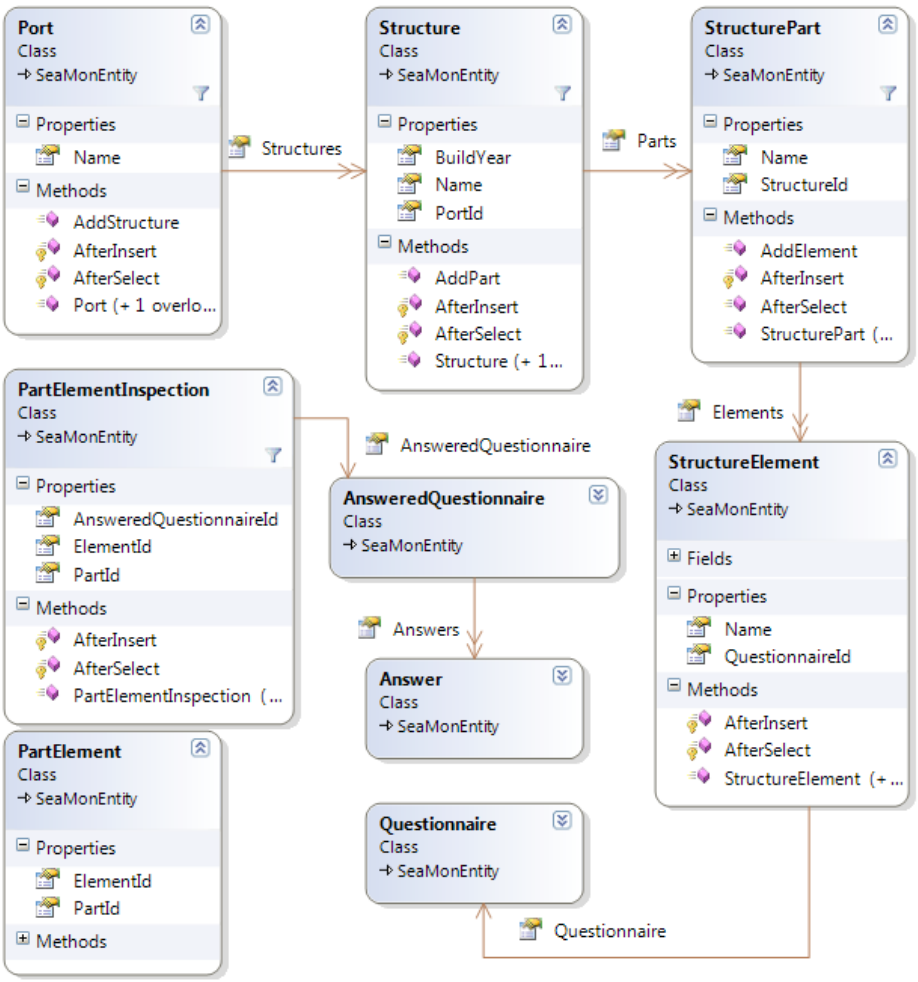

Figura 43 - Diagrama de classes do modelo de domínio da aplicação móvel

3.3.3. UI

O módulo UI fornece uma implementação genérica de uma View e Controller do modelo MVC e serve o mesmo propósito do módulo idêntico criado na aplicação *desktop*. Na figura 44, que apresenta o diagrama de classes deste módulo, pode observar-se a classe `MainForm` que define a janela principal da aplicação e que implementa a interface `IViewManager`. Esta classe serve de:

- Contentor para as Views da aplicação, cujas implementações derivam da classe `Panel` do *namespace* `System.Windows.Forms`;
- Gestor da navegação entre Views. Utiliza internamente uma pilha que armazena o "caminho" percorrido pelo utilizador até à View actualmente a ser apresentada, permitindo voltar atrás nesse "caminho", ou seja, voltar a apresentar a View anteriormente visualizada.

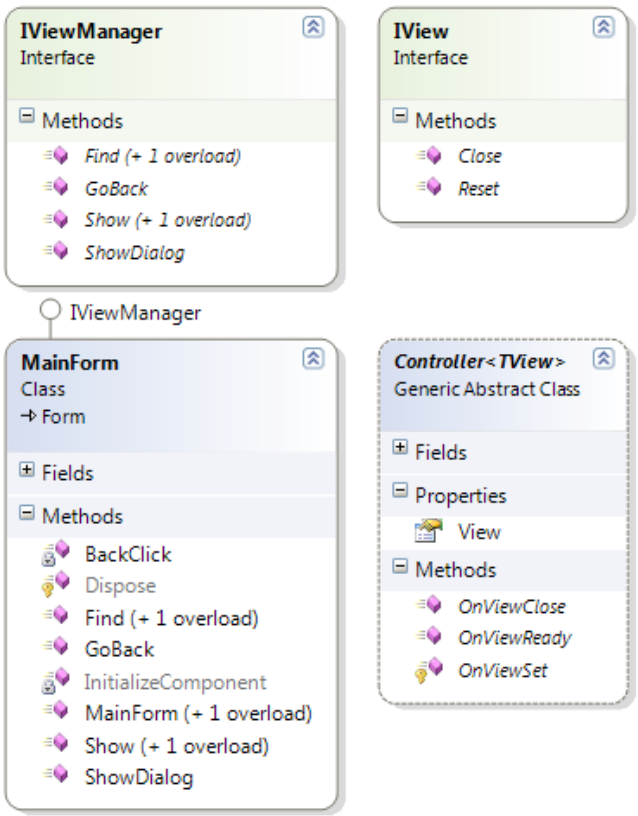

Figura 44 -Diagrama de classes da implementação do MVC

Com base nesta estrutura desenvolveram-se as Views da aplicação que, de um modo geral, se organizam segundo o fluxo ilustrado na figura 45. A aplicação é composta por 4 Views:

- **Welcome View** – Disponibiliza funcionalidades para obter e consultar os dados sobre as estruturas a inspeccionar e enviar os dados das fichas de inspecção preenchidas;
- **Ports View** – Permite visualizar os portos e respectivas estruturas a inspeccionar;
- **Parts View** – Apresenta as partes e elementos estruturais constituintes de uma estrutura previamente seleccionada pelo utilizador;
- **Questionnaire View** – Permite realizar o preenchimento de uma ficha de inspecção, apresentando, uma a uma, as perguntas e as respostas possíveis de uma ficha de inspecção.

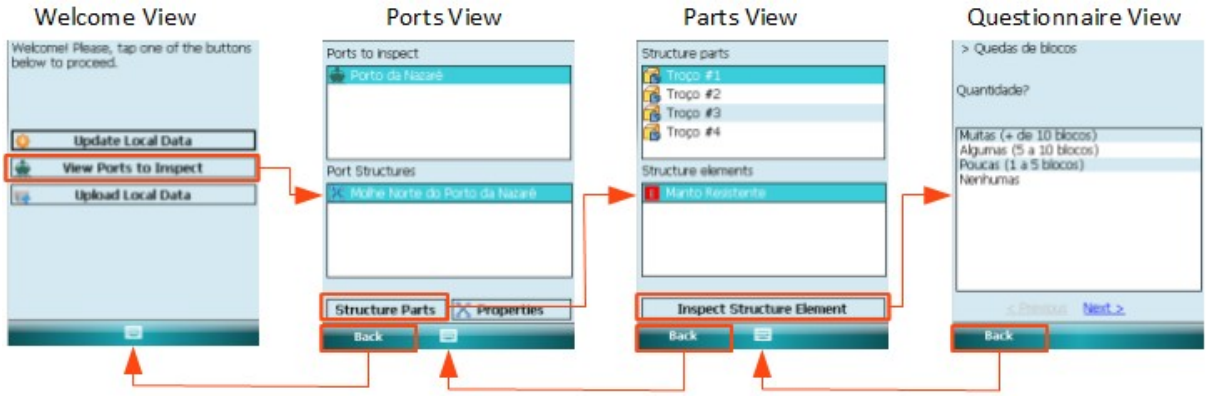

Figura 45 - Fluxo de janelas da aplicação móvel

Cada janela define e implementa uma View, para além do Controller correspondente, tal como é apresentado na figura 46. A implementação das Views é relativamente trivial (a excepção da Questionnaire View), dado que o utilizador não pode editar os seus dados.

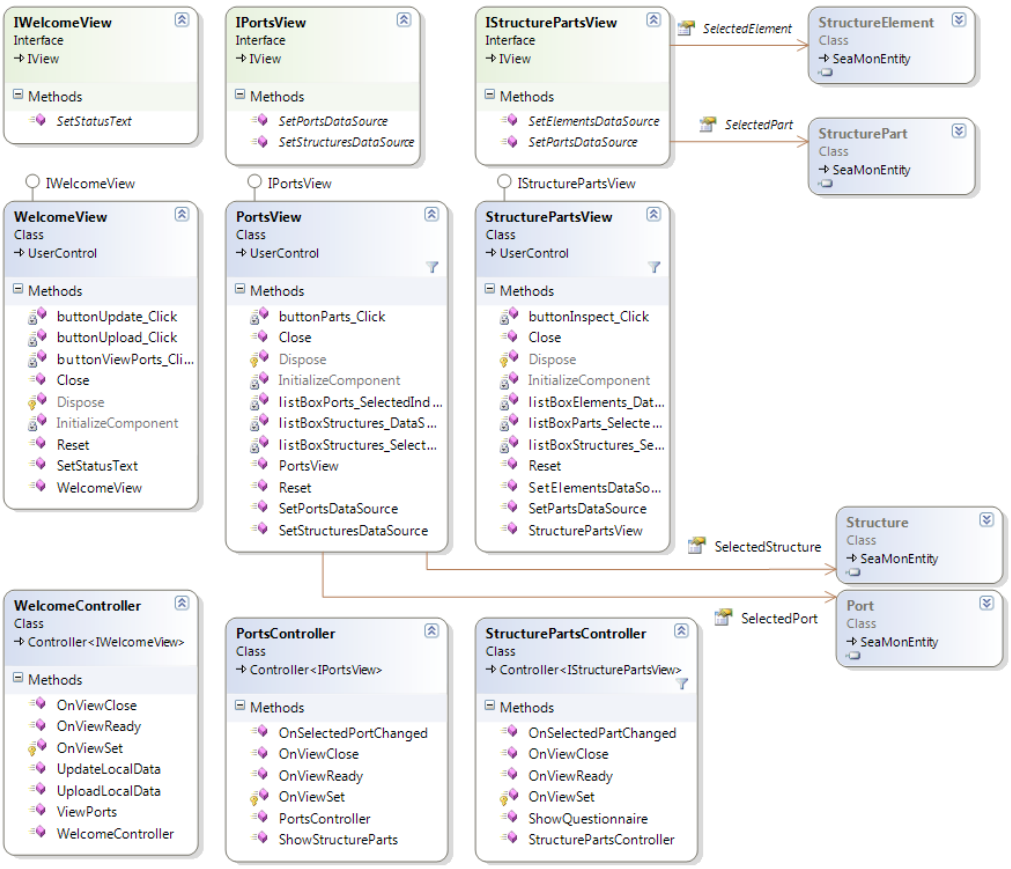

Figura 46 - Diagrama de classes das Views e Controllers da aplicação móvel

A figura 46 não apresenta a `QuestionnaireView` dada a sua complexidade. Representa uma interface gráfica cujos conteúdos são criados dinamicamente com base na definição de uma ficha de inspecção. Por cada pergunta de uma ficha é gerado um `Panel` que permite ao utilizador indicar a sua resposta. O componente a apresentar é determinado pelo tipo de resposta que o utilizador pode dar e está descrito na tabela 14.

Tipo de Resposta	Componente para o preenchimento da resposta
Booleano	Uma `GroupBox` com dois `RadioButtons`
Número	Uma `TextBox` que apenas permite a introdução de dígitos
Texto	Uma `TextBox`
Enumeração	Uma `ListBox` com uma lista dos valores enumerados

Tabela 14 - Mapeamento entre o tipo de resposta e componentes gráficos

O utilizador apenas visualiza uma pergunta de cada vez, mas este comportamento é necessário devido ao tamanho do ecrã que, num dispositivo móvel, é de pequenas dimensões. A figura

47 exemplifica como se pode fazer o particionamento de uma ficha de inspecção em perguntas e, consequentemente, em `Panels`.

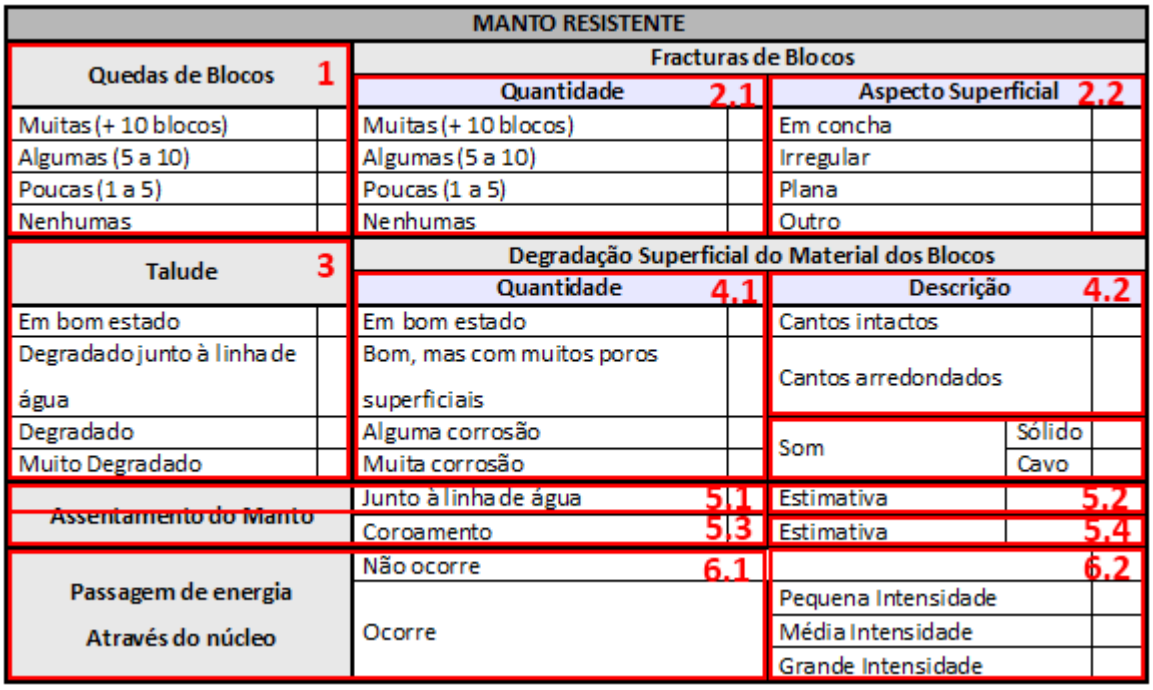

Figura 47 - Particionamento de uma ficha de inspecção em páginas

Dado que a visualização de uma ficha de inspecção é feita de forma "paginada", em que cada "página" é representada por uma pergunta e as suas respostas possíveis, torna-se, por isso, necessário criar mecanismos que permitam navegar entre as várias perguntas do questionário. Uma observação atenta à classe `QuestionnaireView`, do diagrama de classes da figura 48, permite verificar que esta contém métodos que permitem avançar ou retroceder na pergunta actual, assim como visualizar a primeira e última pergunta. De realçar também a interface `IQuestionnaireUIFactory`, que define o método `BuildUI()`, responsável pela criação de uma lista de objectos do tipo `QuestionnaireItemUI` que representam visualmente as perguntas. A *factory* consegue criar estes componentes com base na propriedade `TypeOfAnswer` da classe `Question`, apresentada na figura 37 da secção 3.2.1., que indica o tipo de resposta da pergunta em causa.

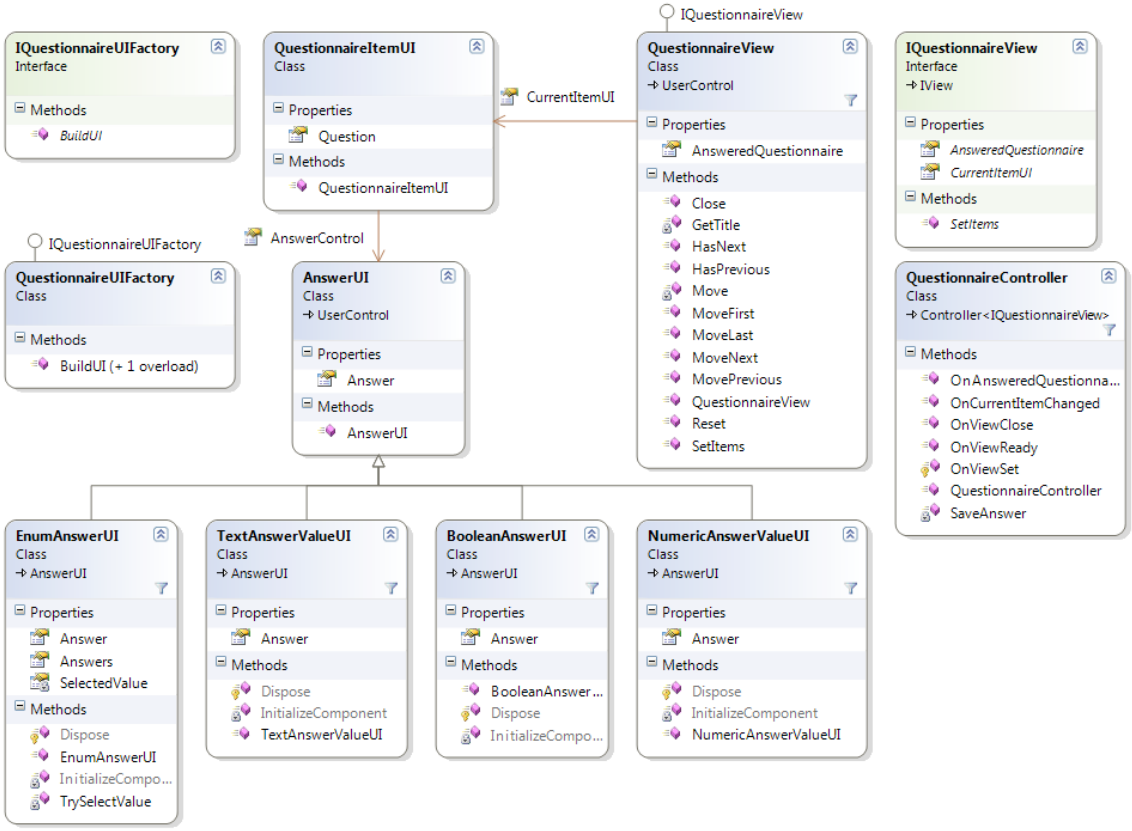

Figura 48 - Diagrama de classes da interface gráfica das fichas de inspecção

A figura 49 mostra um exemplo de como é realizado o preenchimento de uma ficha de inspecção recorrendo à aplicação móvel e, como se pode constatar, a apresentação das perguntas e respostas possíveis é consistente com a definição da ficha de inspecção ilustrada na figura 47. As respostas dadas a cada pergunta são armazenadas automaticamente na BD local, à medida que o utilizador vai assinalando a resposta e navegando pelas perguntas da ficha. Este automatismo foi concebido de modo a facilitar o preenchimento da ficha, dado haver a possibilidade de uma inspecção ser efectuada sob condições climatéricas austeras.

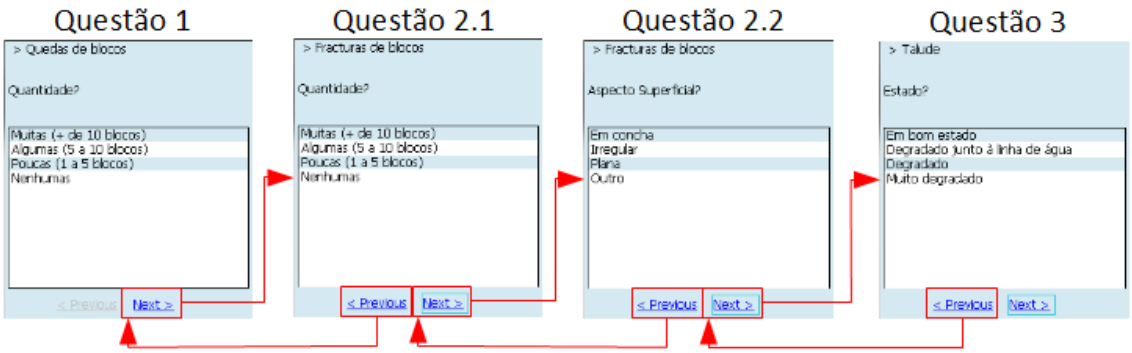

Figura 49 - Exemplo do preenchimento de uma ficha de inspecção

3.3.4. Services

Os serviços para a aplicação móvel estão relacionados com a comunicação entre esta e o servidor e sincronização de dados da BD local. A figura 50 apresenta o diagrama de classes dos três serviços que foram implementados:

- **Remote Service** – Representa o cliente do Web Service. Define dois métodos para obtenção e envio dos dados. O modelo destes dados encontra-se descrito no capítulo 3.2.1.;
- **Entity Translator Service** – Implementação do padrão de desenho Entity Translator, também descrito no capítulo 3.2.1.;
- **Port Service** – Coordena os serviços anteriormente referidos, a fim de proceder à sincronização de dados entre a aplicação móvel e o servidor. Permite também obter a lista de portos e restantes dados associados a partir da BD local.

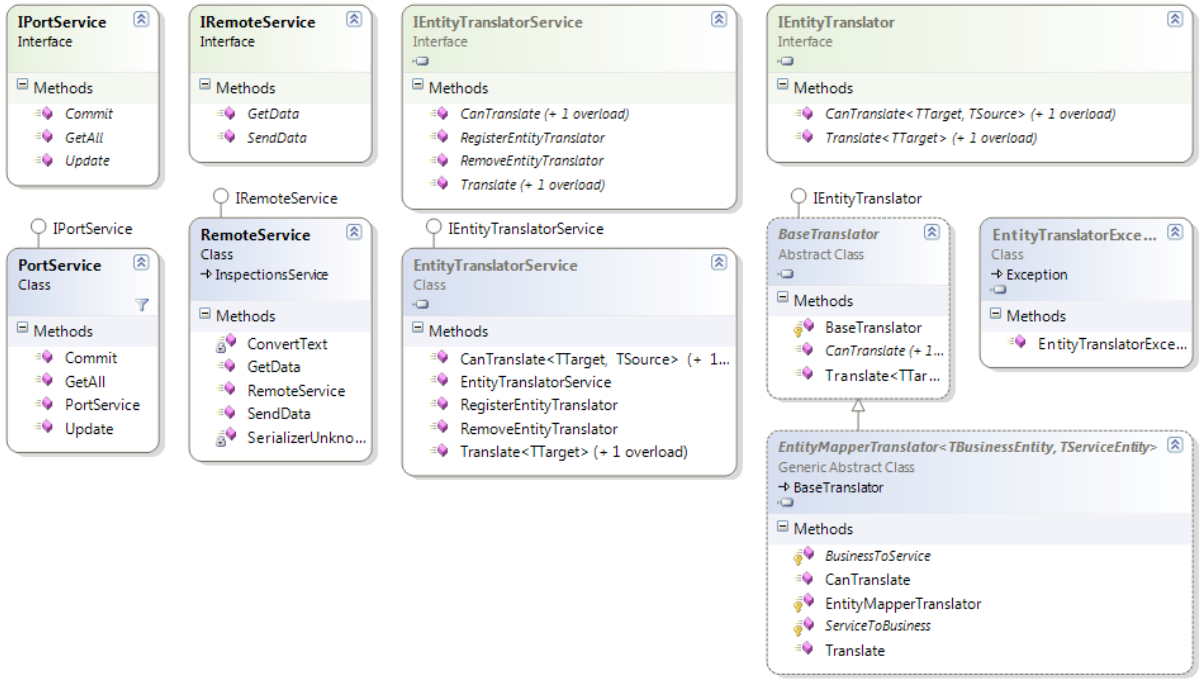

Figura 50 - Diagrama de classes dos serviços da aplicação móvel

A obtenção dos dados das obras a inspeccionar é obtido invocando o método `getInspectionData()` do Web Service da aplicação servidor, acção que pode ser

executada a partir da janela relativa à `WelcomeView`. Após obtenção do resultado, a aplicação móvel procede da seguinte forma:

1. Invoca o Serviço de Tradução de entidades para converter a `InspectionMessage` obtida do servidor para uma `ModelMessage`;

2. Invoca o serviço de base de dados, representado pela interface `IDatabase`, para persistir os dados dessa mensagem.

O diagrama de sequência da figura 51 ilustra a interacção descrita nos pontos referidos anteriormente.

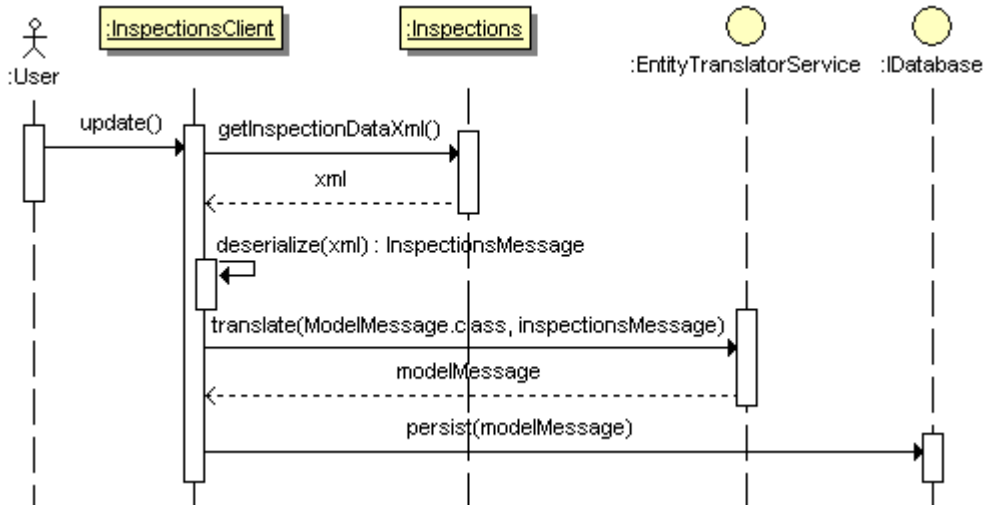

Figura 51 - Diagrama de sequência da persistência de dados na aplicação cliente

Se processo descrito anteriormente for executado com sucesso, então pode assumir-se que a aplicação cliente tem armazenado na sua BD local todos os dados necessários para a realização de inspecções, não sendo, por isso, necessário haver mais troca de mensagens com o servidor. Desta forma, reduz-se os gastos associados ao envio de dados pela rede, sendo apenas necessário efectuar novas comunicações quando o utilizador pretender enviar as fichas de inspecção preenchidas. O processo de envio das fichas preenchidas pode ser invocado a partir da janela relativa à `WelcomeView` e sucede da seguinte forma:

1. Obtém todas as fichas de inspecção preenchidas que se encontram armazenadas na base de dados da aplicação cliente;

2. Cria uma nova `ModelMessage` e atribui-lhe a lista de fichas obtida no ponto anterior;

3. Invoca o Serviço de Tradução de entidades para traduzir de uma `ModelMessage` para uma `InspectionMessage`;

4. Envia a `InspectionMessage` para o servidor, através da invocação do método `saveInspections()` do Web Service.

O diagrama de sequência da figura 52 descreve graficamente os passos enumerados anteriormente.

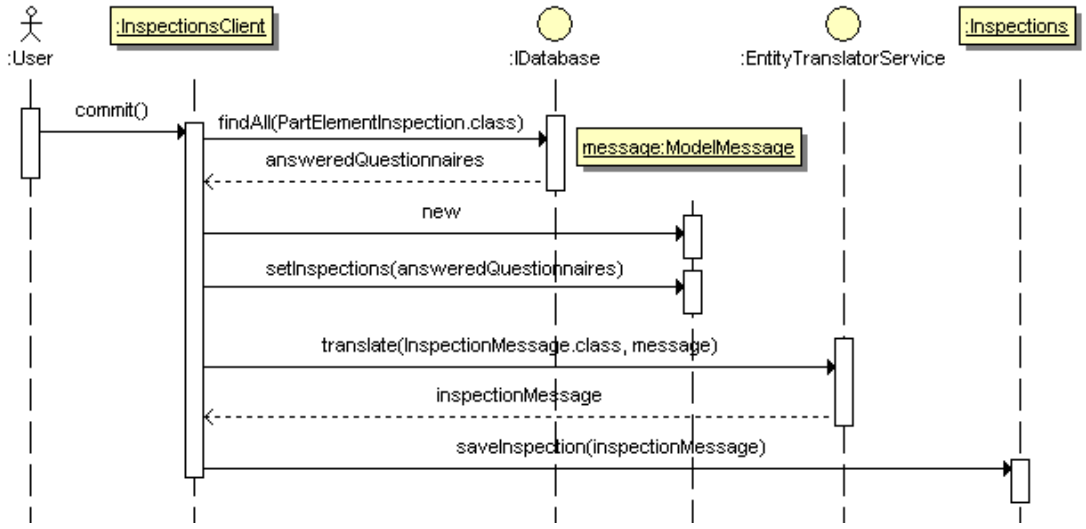

Figura 52 - Diagrama de sequência do processo de envio de dados

Esta página foi intencionalmente deixada em branco

4. Conclusão

Este documento apresentou o trabalho realizado no âmbito deste projecto, cujo tema diz respeito à realização de inspecções visuais a obras marítimas. Foi efectuado o estado de arte relativamente aos procedimentos que são levados a cabo para a realização de uma inspecção, o que implicou recolher dados sobre a caracterização de obras marítimas, assim como obter exemplos de algumas fichas de inspecção, a partir das quais foi possível identificar padrões respeitantes à sua criação e sistematizar a definição de uma ficha. Foi também efectuado o levantamento de informação relativamente a TIs com relevância para o tema em análise e, com base neste estudo, foram escolhidas as linguagens e bibliotecas que permitiram implementar um sistema informático, composto por três aplicações desenvolvidas segundo a arquitectura DDD:

- Uma aplicação *desktop* para definição de fichas de inspecção e visualização das fichas preenchidas;
- Uma aplicação móvel para preenchimento de fichas de inspecção;
- Uma aplicação servidor para sincronização de dados entre a aplicação *desktop* e móvel.

O estado de arte teve grande importância para o sucesso deste projecto. Muitas vezes, um informático tem a tendência a ganhar especialização num número reduzido de linguagens e, normalmente, as suas decisões em relação a TIs são afectadas por essas linguagens. No entanto, é necessário ter em consideração que uma linguagem é uma ferramenta, e, como qualquer outra ferramenta, umas são mais adequadas que outras para a realização de

determinados trabalhos. O levantamento de informação realizado permitiu determinar que a plataforma JAVA era a mais adequada para a implementação das aplicações *desktop* e servidor, devido ao facto de existirem muitas bibliotecas e plataformas aplicacionais que complementam e acrescentam funcionalidades à própria plataforma JAVA, para além de serem gratuitas e de estarem desenvolvidas sob elevados padrões de qualidade. Neste caso em concreto, há que realçar que a aplicação *desktop* e a aplicação servidor são suportadas pelas plataformas NetBeans e Glassfish respectivamente, e que desempenharam um papel fundamental para a criação deste sistema, fornecendo soluções *out-of-the-box* que reduzem o volume de trabalho e tempo necessário para a sua implementação.

O estado de arte permitiu também concluir que a tecnologia a usar para o desenvolvimento da aplicação móvel deveria suportar sistemas operativos Windows Mobile (neste caso em particular, recorrendo à .NET Compact Framework), dado que a maioria dos fabricantes de dispositivos móveis empresariais encontrados utilizam este sistema operativo, sendo esta a gama de produtos que mais provavelmente irá ser utilizada, dado que a realização de inspecções poderá exigir que um dispositivo proporcione outras qualidades no que diz respeito a resistência a quedas, resistência à água, etc.

Outro aspecto que teve um papel preponderante no desenvolvimento do sistema tem a ver com a aplicação de padrões de desenho para a definição e implementação da sua arquitectura, baseada em DDD. Um padrão de desenho é pensado para fornecer a solução mais adequada para a resolução de um problema comum no desenvolvimento de sistemas informáticos e a sua aplicação garante resultados previsíveis e optimizados. Tendo em conta que o DDD é definido através da composição de um conjunto de padrões de desenho que se integram e cooperam entre si, pode considerar-se que esta arquitectura define uma solução genérica para o desenvolvimento de aplicações de dados implementadas segundo o paradigma da programação orientada a objectos. O esforço inicial para criar um sistema baseado neste tipo de arquitecturas é grande e poderá colocar-se em causa a sua utilização para aplicações de dimensões mais reduzidas. No entanto, a partir de uma infra-estrutura desta natureza, foi possível criar um sistema que:

- Contém um conjunto de funcionalidades comuns que podem ser utilizadas numa vasta gama de aplicações;
- Simplifica a adição de novos módulos e funcionalidades;
- Simplifica a alteração de funcionalidades existentes;
- Converge grande parte dos esforços de desenvolvimento no modelo de domínio, catalisando a definição de modelos expressivos e reutilizáveis. O modelo de inspecções concebido neste projecto comprova este aspecto – permite criar, de forma flexível e genérica, um conjunto de fichas de inspecção e associá-las a estruturas marítimas de qualquer tipo, enquanto que, por exemplo, a aplicação ANOSOM tem a limitação de apenas gerir inspecções a quebra-mares.

Dado que esta arquitectura promove a utilização de padrões de desenho e a separação de responsabilidades em camadas distintas, foi possível reaproveitar módulos e modelos da aplicação *desktop* na aplicação servidor, o que efectivamente proporcionou também uma redução no volume de trabalho e tempo necessário para a implementação deste sistema.

Por último, resta falar sobre a gestão de dados geográficos, uma área do sistema que peca por defeito no que toca às funcionalidades implementadas. Investiu-se bastante tempo a investigar sobre tecnologias SIG numa tentativa de apetrechar a aplicação móvel com capacidades que permitissem renderizar uma obra marítima e os seus troços constituintes no ecrã, indicando o troço da obra onde o utilizador se localiza, recorrendo para isso à leitura de dados provenientes do receptor GPS do dispositivo móvel. Esta e outras funcionalidades relativas a este requisito foram deixadas para segundo plano dada a complexidade da sua implementação, pois os sistemas operativos móveis disponibilizam um conjunto reduzido de capacidades gráficas, em comparação com sistemas operativos para ambientes *desktop* (por exemplo, a .NET Compact Framework não disponibiliza funcionalidades para realizar transformações). Por outro lado, as funcionalidades que se encontram já disponíveis para a gestão de dados geográficos (ler e armazenar geometrias a partir de uma BD, suporte para um conjunto de SGBDs sem haver a necessidade de alterar o código da aplicação) servirão de base para desenvolvimentos futuros.

4.1. Desenvolvimentos futuros

Os desenvolvimentos futuros a curto prazo dizem respeito a aspectos que não foram tidos em conta ou que não foram implementados neste projecto:

- **Criar versões de fichas de inspecção** – A alteração da definição de uma ficha de inspecção não pode ser realizada caso já existam fichas de inspecção preenchidas. De modo a contornar esta limitação, o sistema deve permitir criar versões de fichas de inspecção, mantendo inalterados os dados das versões antigas.

- **Avaliar uma ficha de inspecção** – A avaliação pode ser calculada cruzando os valores atribuídos a cada resposta possível no momento da definição de uma ficha de inspecção e as respostas dadas pelo inspector. No entanto, esta tarefa depende de uma sistematização a ser levada a cabo por especialistas na área;

A médio prazo, podem identificar-se os seguintes requisitos:

- Renderização de geometrias, tanto na aplicação *desktop*, como na aplicação móvel;

- Recolha e integração de dados sobre o clima de agitação marítima ao qual uma obra marítima está sujeita;

- Evoluir o modelo para caracterização de uma obra marítima, de modo a, por exemplo, incorporar dados sobre os materiais que a constituem;

- Evoluir o modelo de inspecções de forma a abranger outras estruturas de um porto e activos de zonas costeiras (praias, qualidade da água, etc).

Bibliografia

1: João A. Santos et al, *Novos Instrumentos para a Inspecção e Diagnóstico de Quebra-mares de Taludes*, 2005

2: Rute Lemos; João A. Santos, *ANOSOM - Análise da Observação Sistemática de Obras Marítimas*, 2007

3: Boston GIS, *Cross Compare SQL Server 2008 Spatial, PostgreSQL/PostGIS 1.3-1.4, MySQL 5-6*, 2009, http://www.bostongis.com/?content_name=sqlserver2008_postgis_mysql_compare

4: Boston GIS, *Compare SQL Server 2008 R2, Oracle 11G R2, PostgreSQL/PostGIS 1.5 Spatial Features*, 2009, http://www.bostongis.com/?content_name=sqlserver2008r2_oracle11gr2_postgis15_compare

5: Christian Bauer, Gavin King, *Hibernate in Action*, 2004, Manning

6: Karel Maesen, *Hibernate Spatial*, 2010, http://www.hibernatespatial.org/

7: Vivid Solutions, *Java Topology Suite*, 2010, http://www.vividsolutions.com/JTS/JTSHome.htm

8: Diego Guidi, *NetTopologySuite*, 2009, http://code.google.com/p/nettopologysuite/

9: OpenStreetMap, *OpenStreetMap Wiki*, 2010, http://wiki.openstreetmap.org/wiki/Main_Page

10: Canalys, *Canalys Q3 2009 Report*, 2009, http://www.canalys.com/pr/2009/r2009112.html

11: Deepak Alur, John Crupi, Dan Malks, *Core J2EE Patterns: Best Practices and Design Principles, Second Edition*, 2003, Prentice Hall

12: Antonio Goncalves, *Beginning Java EE 6 Platform with GlassFish 3, 2nd Edition*, 2010, Apress

13: Eric Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*, 2003, Addison-Wesley

14: Martin Fowler, *Inversion of Control Containers and the Dependency Injection Pattern*, 2004, http://martinfowler.com/articles/injection.html

15: Martin Fowler, *Patterns of Enterprise Application Architecture*, 2002, Addison-Wesley

16: Ramnivas Laddad, *AspectJ in Action, 2nd Edition*, 2010, Manning

17: Gregor Hohpe, Bobby Wolf, *Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions*, 2003, Addison-Wesley

18: Frank Krueger, *SQLite-Net*, 2010, http://code.google.com/p/sqlite-net/

19: OraFAQ, *Oracle 7 - Oracle FAQ*, 2009, http://www.orafaq.com/wiki/Oracle_7

20: Refractions, *PostGIS History*, 2008, http://www.refractions.net/products/postgis/history/

21: Microsoft Corporation, *SQL Server 2008 R2 Pricing*, 2008, http://www.microsoft.com/sqlserver/2008/en/us/pricing.aspx

22: Oracle, *Oracle Technology Global Price List*, 2008, www.oracle.com/corporate/pricing/technology-price-list.pdf

Anexos

Anexo A – Critérios de classificação de danos

A.1 – Critérios de classificação de danos de estruturas em talude

Danos		**Comportamento**			**Risco de Ruína**	**Programação**	
Grau	**Estado**	**Estrutural**		**Funcional**		**Inspecções**	**Intervenções**
		Talude	**Vertical**	**Talude/Vertical**			
0	**Bom**	- Não há quedas; - Não há fracturas; - Talude em bom estado; - Material em bom estado.	- Não há fracturas; - Não há corrosão; - Não existe infraescavação, assentamento, deslizamento ou derrubamento.	- Não há galgamentos com intensidade suficiente para afectar a área protegida pela estrutura; - Não há passagem de energia através do corpo da estrutura;	**Praticamente nulo**	Inspecção de rotina	
1	**Danos ligeiros**	Verifica-se um ou mais dos seguintes casos, que não são excedidos: - Quedas: Poucas (1 a 5); - Fracturas : Poucas (1 a 5); - Talude degradado junto à água; - Corrosão: Bom, mas com muitos poros superficiais.	Verifica-se 1 ou mais dos seguintes casos, que não são excedidos: - Fracturas: Poucas, de pequena extensão; - Corrosão pequena; - Apenas se verifica um dos movimentos, em extensão inferior a 50 cm.	Verifica-se 1 ou mais dos seguintes casos, mas que não são excedidos: - Galgamentos ocasionais com intensidade suficiente para afectar a integridade ou operacionalidade da área protegida; - Ligeira passagem de energia através do corpo da estrutura.	**Baixo**	Inspecção de rotina	Manutenção de rotina
2	**Danos médios**	Verifica-se um ou mais dos seguintes casos, que não são excedidos: - Quedas: Algumas (6 a 10); - Fracturas: Algumas (6 a 10); - Talude degradado; - Alguma corrosão;	Verifica-se 1 ou mais dos seguintes casos, que não são excedidos: - Fracturas: Algumas, de continuidade média; - Corrosão média; - Infraescavação, assentamento e deslizamento de amplitude inferior ou igual a 50 cm em extensão não superior a 5 metros; - Rotação por derrubamento não superior a 10º;	Verifica-se 1 ou mais dos seguintes casos, mas que não são excedidos: - Danos estruturais que afectam moderamente a funcionalidade da estrutura; - Galgamentos frequentes de intensidade moderada mas suficiente para afectar a integridade ou operacionalidade da área protegida; - Passagem moderada de energia através do corpo da estrutura.	**Médio**	Inspecção adicional no final do presente Inverno e/ou antes do próximo Inverno	Programar reparação
3	**Danos elevados**	Verifica-se pelo menos um dos seguintes casos: - Quedas: (mais de 10); - Fracturas : Muitas (mais de 10); - Talude muito degradado; - Muita corrosão;	Verifica-se pelo menos 1 dos seguintes casos: - Fracturas: Muitas, de continuidade igual ou superior a média; - Grande corrosão; - Infraescavação, assentamento e deslizamento de amplitude superior a 50 cm em extensão igual ou superior a 5 metros; - Rotação por derrubamento superior a 10º.	A estrutura está inoperacional devido: - Danos elevados que anularam a funcionalidade da estrutura; - Galgamentos muito frequentes e violentos afectando seriamente a integridade ou operacionalidade da área protegida; - Passagem de energia forte através do corpo da estrutura.	**Elevado**	Inspecção mensal	Reparação urgente

A.2 – Critérios de classificação de danos de paramentos

Danos		Comportamento		Risco de Ruína	Programação	
Grau	**Estado**	**Estrutural**	**Materiais**		**Inspecções**	**Intervenções**
0	**Bom**	Não são detectáveis: - Infraescavações ao nível da fundação; - Assentamentos localizados ou em extensão; - Deslizamentos; - Derrubamentos.	Não são detectáveis degradações dos paramentos aparentes, nomeadamente: - Fracturas; - Falta de recobrimento de armaduras; - Corrosão de armaduras;	**Praticamente nulo**	Inspecção de rotina (anual)	
1	**Danos médios**	Verifica-se um ou mais dos seguintes casos: - Infraescavações ao nível da fundação em zonas restritas e localizadas; - Assentamentos pontuais e localizados; - Rotação da estrutura em zonas pontuais e não extensas.	Verifica-se um ou mais dos seguintes casos: - Fracturas e/ou fendas: poucas e de pequena extensão; - Falta de recobrimento de armaduras em zonas restritas e localizadas; - Corrosão de armaduras em zonas restritas e localizadas; - Degradação ligeira do betão na zona sujeita à maré;	**Médio**	Inspecção anual complementada com: - Levantamento topográfico da estrutura; - Monitorização dos assentamentos e dos deslocamentos da estrutura;	Manutenção de rotina e Programação de reparação
2	**Danos elevados**	Verifica-se pelo menos um dos seguintes casos: - Quedas: (mais de 10); - Fracturas : Muitas (mais de 10); - Talude muito degradado; - Muita corrosão;	A estrutura está inoperacional devido: - Danos elevados que anularam a funcionalidade da estrutura; - Galgamentos muito frequentes e violentos afectando seriamente a integridade ou operacionalidade da área protegida; - Passagem de energia forte através do corpo da estrutura.	**Elevado**	Inspecção mensal complementada com: - Levantamento topográfico da estrutura; - Monitorização dos assentamentos e dos deslocamentos da estrutura;	Reparação urgente

Anexo B – Comparação de SGBDs

B.1 – Comparação de características gerais de SGBDs

Aspecto	SQL Server 2008	Oracle 11G R2	PostgreSQL 8.4 (PostGIS 1.5)	MySQL 5.1
SO's suportados	Windows XP+, Windows 2003+ (32 e 64 bits)	Windows 2003+, Linux, Unix (32 e 64 bits)	Windows 2000+ (32 bits apenas), Linux, Unix e Mac (32 e 64 bits)	Windows XP+, Linux, Unix e Mac
Licenciamento	Comercial de código fechado. A versão Express (gratuita) suporta todas as funcionalidades espaciais, embora estabeleça limitações no tamanho da BD e apenas usa um processador	Comercial de código fechado. A versão Express (gratuita) inclui o Oracle Locator, uma versão limitada do componente Oracle Spatial (o componente que disponibiliza funcionalidades espaciais na versão Enterprise)	PostgreSQL tem uma licença BSD, o PostGIS tem uma licença GPL. Pode ser usado em aplicações comerciais, desde que não se façam alterações no código	Comercial de código aberto. Alguns dos componentes incluídos têm licença GPL.
Data da 1ª versão	2008	1996 [19]	2001 [20]	2005
Ferramentas gratuitas para carregamento de dados geográficos	Shape2SQL	shp2sdo (incluído na distribuição do Oracle), OGR2OGR, GeoKettle	Shp2pgsql, shp2pgsql-gui (incluídos na distribuição do PostGIS), QuantumGIS, GeoKettle, Spatial Data Integrator	OGR2OGR, scripts SQL (shp2mysql.sql)
Ferramentas comerciais para carregamento de dados geográficos	Safe FME Objects, ESRI ArcGIS 9.3+, CadCorp	Safe FME Objects, ESRI ArcGIS	Safe FME Objects, ESRI ArcGIS 9.3+, CadCorp	Manifold, FME Objects, ESRI ArcGIS 9.3
Versão e preço	**Express** – gratuito (limitado a 1 processador e BD's até 10 GB) **Standard** – a partir de USD $7500/CPU **Enterprise** – a partir de USD $29000/CPU **Datacenter** – a partir de USD $58000 **Parallel Data Warehouse** – a partir de $58000/CPU [21]	**Express** – gratuito (limitado a 1 processador, BD's até 4 GB e 1 GB de memória RAM) **Personal Edition** – a partir de USD $450/utilizador **Standard Edition One** – a partir de USD $5800/CPU **Standard Edition** – USD $17500/CPU **Enterprise Edition** – a partir de USD $47500/CPU. A opção Oracle Spatial custa USD $17500/CPU [22]	Gratuito. Não tem esquema de versões.	Community Server – gratuito Enterprise Server (Unlimited) – a partir de USD $40000
Bibliotecas de acesso a dados geográficos	ADO.NET 3.5+ (incluída na .NET Framework), SharpMap.NET, NetTopologySuite	SDO API (implementada em Java e incluída na distribuição Oracle), OGR/GDAL, SharpMap.NET	SharpMap.NET, postgis.jar (incluído com o PostGIS), dezenas de bibliotecas em Java (p.e. Java Topology Suite), GDAL C++, AutoCAD FDO	GDAL C++, SharpMap via OGR, AutoCAD FDO
Bibliotecas de ORM gratuitas	HibernateSpatial e NHibernateSpatial	HibernateSpatial	NHibernateSpatial e HibernateSpatial	NhibernateSpatial e HibernateSpatial
Visualizadores e editores gratuitos	Integrado no SQL Manager (apenas para visualização)	GvSig, QuantumGIS, OpenJump, uDig, GeoRaptor para o Oracle SQL Developer	OpenJump, QuantumGIS, GvSig, uDig	GvSig
Visualizadores e editores comerciais	ESRI ArcGIS 9.3 Server SDE, Manifold, CadCorp, AutoCAD FDO, MapInfo 10+	ESRI ArcGIS SDE, FME, Manifold, CadCorp, AutoCAD FDO, MapInfo	ESRI ArcGIS 9.3 Server, ZigGIS for ArcGIS/ArcMap Desktop, Manifold, FME, CadCorp, AutoCAD FDO	FME
Funções espaciais	Implementa as especificações do OGC SFSQL MM and Geodetic custom (mais de 70 funções)	Implementa funções geodésicas e planares (2D), embora o Oracle Locator não forneça funções como intersecções e uniões.	Mais de 300 funções e operadores primordialmente focadas em cálculos de proximidade	Implementa a especificação do OGC apenas para 2D
Índices espaciais	Uma adaptação de B-Tree	R-Tree	GIST – uma variação de R-Tree	R-Tree
Suporte geodésico (cálculo de medições sobre um sistema de coordenadas esférico)	Sim, mas com algumas restrições	Sim	Sim, mas com algumas restrições	Não

B.2 – Funcionalidades espaciais de SGBDs

Funcionalidade	SQL Server 2008	Oracle 11G R2	PostgreSQL 8.4/PostGIS 1.5	MySQL 5.1
Tipos de geometrias suportadas	Polygon, Point, LineString, MultiLineString, MultiPoint, MultiPolygon, GeometryCollection 2D Suporte para 3D e 4D	Todos os tipos básicos definidos pelo OGC para 2D, 2.5D e 3D (suporta volumetria)	Todos os tipos básicos definidos pelo OGC para 2D, 2.5D	Geometrias 2D Suporta 3D e 4D para armazenamento, mas não para processamento – Polygon, Point, LineString, MultiPoint, MultiPolygon, MultiLineString, GeometryCollection
Transformações	Não. É necessária a utilização de ferramentas de outros ISV's.	Sim, através da função SDO_CS.TRANSFORM	Sim, através da função ST_Transform	Sim, através da função ST_Transform
Funções para exportação de geometrias	STAsBinary, STAsText, AsGML, AsTextZM	GET_WKB, GET_WKT, Package SDO_UTIL contém: TO_WKBGEOMETRY, TO_WKTGEOMETRY, TO_GML311GEOMETRY, TO_GMLGEOMETRY, TO_KMLGEOMETRY	ST_AsBinary, ST_AsText, ST_AsSVG, ST_AsGML, ST_AsKML, ST_AsGeoJson, ST_AsEWKT, ST_AsHexEWKB	AsBinary(), AsText()
Funções para importação de geometrias	STGeomFromText, STGeomFromWKB, GeomFromGML	SDO_Geometry importa dados em formato binário e em WKT, a package SDO_UTIL contém: FROM_GML311GEOMETRY, FROM_GMLGEOMETRY, FROM_KMLGEOMETRY	ST_GeomFromText, ST_GeomFromWKB, ST_GeomFromGML, ST_GeomFromKML	GeomFromText(), GeomFromWKB()
Intersecções, Diferenças, Uniões	STIntersects, STIntersection, ST_Union, STSymDifference	SDO_DIFFERENCE, SDO_INTERSECTION, SDO_UNION	ST_Intersects, ST_Intersection, ST_Union, ST_Difference, ST_SymDifference	MBRIntersects()
Outras funções de relação	STContains, STDifference, STDisjoint, STEquals, STIntersects, Filter, STOverlaps, STRelate, STSymDifference, STTouches, STWithin	SDO_CONTAINS, SDO_COVEREDBY, SDO_COVERS, SDO_EQUAL, SDO_FILTER, SDO_INSIDE, SDO_JOIN, SDO_NN, SDO_NN_DISTANCE, SDO_ON, SDO_OVERLAPBYDISJOINT; SDO_OVERLAPBYINTERSECT, SDO_OVERLAPS, SDO_RELATE, SDO_TOUCH, SDO_WITHIN_DISTANCE	ST_Contains, ST_CoveredBy, ST_Covers, ST_Disjoint, ST_Difference, ST_DWithin, ST_Equals, ST_Intersects, ST_Overlaps, ST_Relate, ST_Touches, ST_Within	MBRContains(), MBREqual(), MBROverlaps(), MBRTouches(), MBRWithin(), Contains(), Overlaps(), Equals()
Funções Acessoras, Editoras e Processadoras	BufferWithTolerance, MakeValid, Reduce, STSimplify, STBoundary, STBuffer, STCentroid, STConvexHull, STDimension, STEndPoint, STExteriorRing, STGeometryN, STGeometryType, STInteriorRingN, STIsClosed, STIsEmpty, STIsSimple, STIsValid, STNumGeometries, STNumInteriorRing, STNumPoints, STPointN, STStartPoint, STSRID, STUnion, STX, STY	ST_Centroid, ST_ConvexHull, SDO_BUFFER, SDO_CENTROID, SDO_CONVEXHULL, SDO_POINTONSURFACE, SDO_TRIANGULATE	ST_Affine, ST_Boundary, ST_Buffer, ST_Centroid, ST_ClosestPoint, ST_ConvexHull, ST_Dimension, ST_EndPoint, ST_ExteriorRing, ST_GeometryN, ST_GeometryType, ST_InteriorRingN, ST_IsClosed, ST_IsEmpty, ST_IsRing, ST_IsSimple, ST_IsValid, ST_Length, ST_LongestLine, ST_NumGeometries, ST_NumInteriorRings, ST_NumPoints, ST_NPoints, St_NumInteriorRings, ST_PointN, ST_ShortestLine, ST_Simplify, ST_StartPoint, ST_SRID, ST_Translate, ST_Union, ST_X, ST_Y	Centroid(), Dimension(), EndPoint(), Envelope(), ExteriorRing(), GeometryN(), GeometryType(), InteriorRingN(), IsClosed(), IsRing(), NumPoints(), PointN(), SRID(), StartPoint()
Funções de medição	STArea, STLength, STDistance	ST_Area, ST_Length, ST_Distance com escolha de unidades	ST_Area, ST_Distance, ST_Distance_Spheriod, ST_HausdorffDistance, ST_Length, ST_Perimeter, ST_Length_Spheriod, ST_MaxDistance	Area(), Glength(), Distance()

Anexo C – Lista de exemplos de pares chave/valor predefinidos pelo OpenStreetMap para a definição de Etiquetas

Chave	Valores possíveis
TMC	Definido pelo utilizador
Abutters	residential, retail, commercial, industrial, mixed
access	yes, designated, official, private, permissive, destination, delivery, agricultural, forestry, unknown, no
addr	Definido pelo utilizador
admin_level	Número
aerialway	cable car, chair lift, drag lift, gondola, mixed lift, pylon, station
aeroway	aerodrome, apron, gate, helipad, runway, taxiway, terminal, windsock
amenity	architect office, arts centre, atm, baby hatch, bank, bbq, bench, bicycle parking, bicycle rental, biergarten, brothel, bureau de change, bus station, bus station, cafe, car rental, car sharing, car wash, cinema, clock, college, courthouse, crematorium, dentist, doctors, drinking water, embassy, emergency phone, fast food, fast food, ferry terminal, fire hydrant, fire station, food court, fountain, fuel, grave yard, grit bin, hospital, hunting stand, kindergarten, library, library, marketplace, nightclub, parking, parking, pharmacy, pharmacy, place of worship, place of worship, place of worship, police, post box, post office, prison, pub, public building, recycling, restaurant, restaurant, sauna, school, shelter, stripclub, studio, taxi, telephone, theatre, toilets, townhall, university, vending machine, veterinary, waste basket, waste disposal
barrier	block, bollard, cattle grid, city wall, cycle barrier, ditch, entrance, fence, fence, gate, hedge, kent carriage gate, retaining wall, sally port, stile, toll booth, toll booth, wall